CRIDES ASÍNCRONES EN MPI:
MILLORANT TEMPS I ENERGIA.

Autor: Òscar Camacho Requena

Consultor: Ivan Rodero Castro

TFM - Computació d'Altes Prestacions

Màster Universitari en Enginyeria Informàtica

UOC **Juny 2015**

Agraïments

A la meva família pel seu recolzament continu.
Al meu consultor Ivan Rodero per la seva ajuda i dedicació.
A la Universitat de Rutgers per deixar-me fer servir el clúster de computació on s'han fet totes les proves d'aquest treball.

Índex de continguts

1 Introducció

1.1 Objectiu del treball

En els últims anys s'han pogut crear clústers de supercomputació, fent servir ordinadors de sobretaula o servidors corporatius interconnectats, de tal manera que l'ús de llibreries com MPI han fet que els càlculs abans restringits als supercomputadors, s'hagin socialitzat i tornat menys elitistes.

Aquestes llibreries de programació, paral·lelitzen i reparteixen els càlculs entre tots els nodes d'un clúster d'ordinadors, d'una manera síncrona. Això fa que es perdi temps i energia en el temps d'espera fins que tots els fils d'execució del càlcul han acabat la seva feina.

L'objectiu d'aquest treball és: demostrar que la transformació de càlculs feta de manera asíncrona, fa que disminueixi els temps de processament i el consum d'energia del clúster.

Es faran dues transformacions de crides síncrones d'MPI a asíncrones (disponibles en MPI a partir de la versió 3) i es presentaran els resultats, així com la seva discussió i conclusions.

1.2 Resum

La computació d'altes prestacions, abans restringida als grans superordinadors, ha pogut sortir d'aquestes gran sales i entrar a la gran majoria de centres d'investigació del món pel fet de poder fer servir ordinador de sobretaula o servidors corporatius interconnectats en xarxa. L'ús de llibreries del tipus MPI en aquests clústers, ha fet que càlculs abans restringits a grans i costoses màquines, s'hagin socialitzat.

Aquestes llibreries de programació, paral·lelitzen i reparteixen els càlculs entre tots els components d'un clúster d'ordinadors, de manera que es pot treballar alhora en una sèrie de càlculs i aprofitar tot el potencial de càlcul dels processadors del clúster.

Ara bé, el fet de fer servir diferents processadors, obliga a tenir sincronitzats els càlculs,de manera que si la línia del programa s'obre en quatre per realitzar un càlcul, hem de tenir cura que tinguem tots quatre resultats abans de poder continuar amb el programa.

Això ho soluciona MPI fent les operacions síncrones, és a dir, cada procés fa el seu càlcul i un cop acaba, pregunta per l'estat de la resta de processos. Fins que tots els processos no obtenen la resposta de que tots han acabat, no es continua amb la execució del programa.

Aquesta sincronia fa que els processos facin treballar contínuament els processadors, ja que estan preguntant com van la resta de fils d'execució amb els càlculs.
Si podem transformar aquests fils d'execució de síncron a asíncron, de manera que evitem aquesta pregunta continua sobre l'estat de la resta de fils d'execució del programa de manera que, un cop un fil a acabat entri en espera una determinada quantitat de temps abans de preguntar l'estat de la resta de fils, aconseguim una millora en el consum energètic del clúster i, fins i tot, una millora en el rendiment dels càlculs.

1.3 Motivació

Normalment, les persones que fan servir càlculs en clústers d'alt rendiment computacional, no tenen els coneixements tècnics de programació com per poder optimitzar i millorar els rendiments dels seus càlculs.
Si els oferim la possibilitat de poder accedir a millores només canviant un paràmetre de la línia d'execució del seu programa, els estem facilitant el treball a aquests investigadors.

1.4 Finalitat del treball

Demostraré que es pot optimitzar el temps de càlcul i el consum energètic canviant la filosofia síncrona de la llibreria MPI.
Tornant les barreres síncrones en asíncrones, fent esperar al processadors un cop han acabat la seva part de càlcul i evitant la continua consulta d'estats a la resta de fils, aconseguim optimitzar els recursos computacionals existents.

1.5 Plantejament del problema

S'hauran de trobar exemples de càlculs en MPI que facin servir mètodes síncrons. Un cop escollits, trobar la manera de substituir la barrera síncrona on esperen de manera activa els diferents fils de computació creats, en una barrera asíncrona, on el procés un cop acabat el càlcul continuaria endavant. Haurem de forçar a que quedi en espera (i per tant, sense consumir tanta energia) un determinat espai de temps (definit mitjançant un paràmetre en la línia d'execució) abans de preguntar per l'estat de la resta de fils d'execució del programa. Per totes les execucions es passarà un paràmetre que regularà la complexitat del càlcul del programa. A més, per les variants asíncrones, tindrem un parell de paràmetres més: un que definirà el temps d'espera abans de saber l'estat de l'execució de la resta de fils i saber així si han acabat o no per poder continuar amb l'execució del codi i un segon paràmetre que indicarà el grau de repartiment dels càlculs entre tots els fils. Així doncs, un zero indicarà un repartiment equitatiu de la complexitat dels càlculs entre tots els fils de l'execució del programa i un 100 un grau màxim de desigualtat en aquest repartiment de tasques entre els fils.
La funció d'aquests paràmetres és:

- pel paràmetre de temps d'espera, poder determinar quin és el millor temps d'espera per maximitzar l'execució del programa en funció de temps de computació i energia consumida.
- pel paràmetre de desequilibri, saber si es pot millorar el temps d'execució i l'energia consumida si els fils no executen la mateixa quantitat de càlculs. Potser que tinguem més eficiència si tenim una part dels fils executant més càlculs que una altra part que ja ha fet una part de la feina i resta en espera dels primers.

1.5.1 MPI_Barrier

Aquest mètode serveix per assegurar-nos que cap fil d'execució s'avançarà de la resta en l'execució de codi. En arribar un fil a aquest mètode, restarà aturat fins que no arribin

també la resta de fils d'execució del programa. Moment en el qual, tots continuaran normalment.
És com una barrera física que no s'obre fins que tots els fils han arribat a ella.
En aquest cas, substituïm aquest mètode per MPI_Ibarrier, que es diferencia en que en arribar un fil a aquesta barrera, continua endavant l'execució del codi sense esperar a la resta de fils.
Per evitar que cada fil tingui una execució descontrolada, hem d'afegir un mètode que ens permeti de controlar l'estat de la resta de fils (MPI_Test) per poder abandonar així l'estat latent del procés.

1.5.2 MPI_Reduce

Aquest mètode realitza un càlcul amb els resultats obtinguts per cada fil, reduint-lo a un únic valor. En el nostre cas, fem una suma amb els valors reportats per tots els fils.
En substituir aquest mètode pel seu equivalent asíncron MPI_Ireduce, i com en el cas anterior, per poder saber quan em de treure de l'espera al fil, fem servir el mètode MPI_Test.

1.6 Resum del mètode

Es busca un tipus de programa que realitzi un càlcul complex i susceptible de ser paral·lelitzat. Un cop el tenim es realitza la transformació en el seu funcionament, de síncron a asíncron, de tal manera que un cop un fil ha acabat la seva tasca de càlcul, entre en espera. Cada cert temps, el fil surt de l'espera per saber l'estat de la resta de fils que componen l'execució del programa. Aquest temps es pot definir mitjançant un paràmetre a la línia de comandes d'execució del programa. Un altre paràmetre ens servirà per realitzar el repartiment de tasques entre els fils de l'execució.
S'automatitza mitjançant shellscript el llançament dels programes tant en la seva versió síncrona con asíncrona, modificant els paràmetres de complexitat de càlcul, temps d'espera i desequilibri i guardant els resultats de les execucions en un fitxer de text.
Aquest fitxer de text amb el valors dels paràmetres, el temps d'execució i l'energia consumida, s'importen a una fulla d'excel que permetrà crear les gràfiques que ajudaran a interpretar els resultats obtinguts.

2 Treballs relacionats

2.1 MPI

MPI-3 incorpora instruccions asíncrones o no bloquejants, que tenen l'avantatge de no carregar el sistema amb comunicacions de control entre els fils d'execució d'un procés.
Amb aquesta nova versió, diversos estudis han demostrat que els temps d'execució es veuen reduïts en fer servir aquest tipus de crides en comptes dels mètodes bloquejants tradicionals. Un exemple seria:

Mastering Perfomance Challenges with the New MPI-3 Standard
Mikhail Brinsky, Alexander Supalov, Michael Chuvelev and Evgeny Leksikov
Descàrrega

Hi han d'altres autors que han trobat la manera de solucionar la despesa d'energia en les crides síncrones. En arribar a una barrera bloquejant tenen dues opcions: o fer dormir al processador, o variar el seu voltatge. Tot va encaminat a disminuir el consum d'energia del sistema:

Exploiting Energy Saving Opportunity of Barrier Operation in MPI Programs
Wenjing Yang, Canqun Yang
Second Asia International Conference on Modeling & Simulation, 2008. AICMS 08.
Descàrrega

Aquest treball estaria agafant part de les idees exposades en aquests articles:

- agafant els avantatges de la menor càrrega de comunicacions que donen les crides asíncrones. D'aquesta manera, podem aconseguir millorar els temps d'execució en reduir càrrega de treball.
- En posar el fil d'execució en espera, aconseguim reduir el consum energètic del sistema computacional.

Fusionant aquestes dues idees aconseguim un millor resultat energètic i un menor temps de càlcul. Redundant en un benefici per l'investigador i pel medi ambient.

2.2 Intel

Per una altra banda, Intel amb el seu disseny d'MPI, també aprofita les noves funcionalitats de la versió 3. Tornen a destacar els mètodes no bloquejants d'aquesta nova versió. Un altre tema interessant del qual parlen és la possibilitat de fer un ajust automàtic dels paràmetres d'execució a les aplicacions. Aquest seria un dels punts que han quedat fora de l'abast d'aquest treball, però que té un gran interès com a treball de futur.

Scaling MPI –Hybrid Applications Towards Performance
Michael Steyer
Descàrrega

Standards, Standards & Standards -MPI 3
Michael Steyer
Descàrrega

3 MPI. Mètodes col·lectius síncrons i asíncrons.

Els mètodes col·lectius síncrons possibiliten la comunicació entre un grup de processos, evitant la necessitat de programar protocols de comunicació punt a punt entre processos i sent molt més eficients que aquesta comunicació punt a punt.

Aquest grup de processos es relacionen mitjançant un comunicador intern i tots els seus mètodes seran bloquejants a nivell local.
Dintre del grup de processos tindrem un procés que més endavant anomeno "principal" i que s'encarrega d'aglutinar totes les dades del programa. També s'encarregarà de calcular i distribuir dades així com de rebre les dades calculades per tot els processos del grup definit. Aquesta sincronització dels fils del grup afegeix molts missatges de comunicació entre els fils, que poden arribar a penalitzar l'execució del programa sencer. Per tant, s'ha de fer servir amb cura i no abusar d'aquests mètodes.

Els mètodes col·lectius asíncrons es caracteritzen perquè les seves crides fan retornar immediatament el fil d'execució que el crida. D'aquesta manera, si volem tenir controlada l'execució del programa, hem d'afegir mètodes de sincronització entre fils. Amb tot això, aconseguim que la comunicació entre fils sigui gairebé inexistent, quedant reduïda a la mínima expressió i alliberant recursos del sistema. A priori obtindrem un millor rendiment sense sacrificar res. La pèrdua de la sincronització automàtica dels mètodes síncrons es veu compensada per la no necessitat continua de comunicació entre fils per assegurar aquesta sincronització.

Per realitzat aquest treball es fa servir la versió MPI-3 d'openMPI. Aquesta versió, estandarditzada al novembre de 2014, tot i que encara no és molt utilitzada per la comunitat, incorpora les crides asíncrones necessàries per fer aquest treball.

4 Programes

4.1 Programa de multiplicació de matrius

4.1.1 Presentació del programa

El primer cas del treball recau sobre el mètode de barrera MPI_Barrier. Fem realitzar a cadascun dels fils d'execució uns càlculs i el fem esperar fins que no acabin la resta de fils. En aquest cas, tots els fils realitzen el mateix tipus de càlcul i no depenen dels resultats obtinguts per altres fils.
Un exemple senzill de programa on aplicar aquestes premisses seria una multiplicació de matrius. Determino a 16 el nombre de processos que farà servir el programa.
El programa accepta tres paràmetres d'entrada:

- SIZE: que indica la mida de les matrius a multiplicar. El seu valor per defecte és 100.
- IMBALANCE: on podem fer que cada fil d'execució realitzi més o menys càlculs. D'aquesta manera tindrem fils d'execució que faran més càlculs que uns altres. I aquests últims es veuran forçats a esperar més temps a que acabin aquells fils amb una càrrega computacional més alta. El valor per defecte és 10. Amb aquest paràmetre, juntament amb un valor constant (IMBALANCE_BASE=50) i el número de procés adjudicat per MPI_Comm_rank, es calcula quantes multiplicacions haurà de fer cada procés. Agafarem valors entre 0 i 100. On 0 indica un repartiment equitatiu en les tasques de computació i un 100 un màxim de desigualtat en els càlculs a realitzar per cada procés

- ITERS: és el nombre de vegades que farem servir l'MPI_Barrier. D'aquesta manera podem repetir l'experiment diverses vegades i obtenir una mitjana dels resultats obtinguts. El valor per defecte és 10.

L'execució del programa aniria així:

1) MPI crea els 16 processos i comença la primera volta de càlculs.
2) Segons el rang del procés, haurà de calcular més o menys vegades la multiplicació de dos matrius quadrades de mida SIZE. El procés 0 sempre serà el que menys càlculs realitzarà (degut a l'algoritme fet servir per desequilibrar el treball entre els processos: IMBALANCE_BASE + (IMBALANCE * num_rang_proces).

   Per tant, en el nostre cas de 16 processos i per uns valors d'IMBALANCE de 0, 50 i 100 tindríem un repartiment de tasques com el següent:

num_rang_proces	Nº multiplicacions a fer amb IMBALANCE = 0	Nº multiplicacions a fer amb IMBALANCE = 50	Nº multiplicacions a fer amb IMBALANCE = 100
0	50	50	50
1	50	100	150
2	50	150	250
3	50	200	350
4	50	250	450
5	50	300	550
6	50	350	650
7	50	400	750
8	50	450	850
9	50	500	950
10	50	550	1050
11	50	600	1150
12	50	650	1250
13	50	700	1350
14	50	750	1450
15	50	800	1550

3) Multiplicació de matrius. Es multipliquen dues matrius quadrades de SIZExSIZE, omplertes amb números aleatoris
4) Un cop cada procés a acabat el seu nombre de multiplicacions, s'atura a l'MPI_Barrier i queda a l'espera de la finalització de la resta de processos.
5) Un cop acaben tots els processos els seus càlculs assignats, s'informa per pantalla de l'acabament i es continua amb la següent iteració. El procés s'acabarà un cop fetes les iteracions fixades pel paràmetre ITERS.

4.1.2 Modificacions fetes

Modifiquem el programa de manera que la barrera síncrona, feta servir per aturar els processos més ràpids, la transformem en una barrera asíncrona MPI_Ibarrrier. En aquest cas perd la funcionalitat de barrera, ja que el procés que hi arribi, seguirà amb la execució de la resta de codi del programa. Hem de crear una manera de fer entrar en espera latent el procés perquè no consumeixi energia ni ample de banda preguntant contínuament per

l'estat de la resta de fils. És per això que introduïm un nou paràmetre d'entrada (que s'afegeix als tres ja existents del procés síncron) que permet definir el temps d'espera abans de preguntar per l'estat de la resta de fils d'execució.
Aquest nou paràmetre és el primer en ordre de definició i forma part d'una estructura de temps *timespec*. En concret indicarem els nanosegons (ns) d'espera del procés abans d'executar el mètode MPI_Test. Aquest mètode posarà a *true* un paràmetre de la seva crida quan la resta de fils d'execució hagin acabat els càlculs. Si el resultat de MPI_Test es *false*, el procés entrarà en espera els ns indicats al paràmetre d'entrada.
La resta del programa es comporta i funciona de la mateixa manera que la versió síncrona.

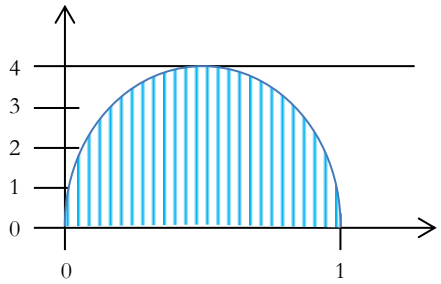
4.2 Càlcul del número π

4.2.1 Presentació del programa

La computació paral·lela pot facilitar molt el càlcul del número π. En aquest cas farem servir pel càlcul la següent equació:

$$\pi = \int_0^1 \frac{4}{1+x^2}\, dx$$

Que podem interpretar com el càlcul de l'àrea sota la corba definida:



Per calcular aquesta integral, podem aplicar el teorema de Riemann, que consisteix en dibuixar un nombre finit de rectangles dintre de l'àrea de la corba a calcular. Calculant la suma de les àrees de tots els rectangles definits, aproximem el valor de l'àrea i, per tant, de la integral. Com més petits siguin aquests rectangles, és a dir, com més rectangles fem servir per realitzar el càlcul, més acurada serà la mesura.

He trobat una adaptació a MPI d'aquest mètode publicat al web de la Washington University in St. Louis. Concretament al seu Center for High Performance Computing (enllaç).
Donada la seva senzillesa i l'ús de la crida síncrona MPI_Reduce, decideixo fer servir aquest codi.

Afegeixo un únic paràmetre d'entrada:

- npts: per poder ajustar el grau de complexitat del càlcul de π indicant el nombre rectangles que s'hauran de fer servir al càlcul.

La primera cosa que fa el programa és repartir el nombre de rectangles a fer servir entre tots els fils d'execució del programa. Després d'això, cada fil escull una coordenada x a l'atzar i càlcul l'àrea del rectangle que li correspon a aquesta coordenada. Després de repetir aquest procés tantes vegades com rectangles li hagin assignat, el fil queda blocat a la crida MPI_Reduce fins que es tenen totes les dades i es pot realitzar la suma global de tots els resultats parcials calculats pels fils d'execució.
Un cop calculada la suma total, s'alliberen els processos i el fil principal mostra per pantalla el valor de π trobat i el nombre de rectangles fets servir.

Per realitzar els càlculs de tots dos programes, he fet servir 16 fils d'execució.

4.2.2 Modificacions fetes

Com en l'anterior programa, he d'afegir dos paràmetres d'entrada a l'existent de la versió síncrona per donar cabuda al temps d'espera dels fils

- tv_nsec, temps d'espera. Amb un valor per defecte de 5000 ns.
- IMBALANCE, per desequilibrar els càlculs entre fils. Amb un valor per defecte de 0 = sense desequilibri.

Una de les modificacions més imaginatives de fer per aquest programa va ser dissenyar el desequilibri de càlculs entre els fils d'execució. D'això s'encarrega el fil principal al principi de tot del programa on la resta de fils queden a l'espera en la crida MPI_Bcast a rebre el valor calculat per cadascun d'ells.
La manera de fer el desequilibri és la següent: es reparteixen equitativament el nombre de rectangles a calcular entre tots els fils d'execució. Si la repartició no pot ser exacta, el sobrant se'l queda el fil principal. Desprès d'això, i anant fil per fil, restem un percentatge dels rectangles assignats. Aquest percentatge és calcula agafant a l'atzar un número entre 0 i el valor del paràmetre IMBALANCE. Aquesta quantitat restada, és sumada a un dels fils d'execució escollit aleatòriament.

Un cop el fil principal té assignat el nombre de càlculs que haurà de fer cada fil, els hi comunica mitjançant la crida MPI_Bcast.
En tenir cada fil d'execució els seus paràmetres de càlcul, passen a realitzar la suma parcial. Quan la tenen calculada, mitjançant la crida asíncrona MPI_Ireduce, es comunica al buffer del fil principal que s'encarrega d'emmagatzemar aquestes sumes parcials. A partir d'aquí, el funcionament és totalment idèntic al programa anterior de multiplicació de matrius asíncron. Preguntem amb MPI_Test l'estat de la resta de fils i si encara tenim fils calculant, entra en espera el temps assenyalat pel paràmetre d'entrada tv_nsec. Un cop transcorregut aquest temps d'espera, torna a preguntar per l'estat de la resta de fils. Entrarà en espera un altre cop si no han acabat o continuarà cap al final del programa si ja han acabat tots.
Un cop tots els fils han acabat els seus càlculs, el fil principal indica per pantalla el valor de π trobat i el nombre de rectangles d'aproximació fets servir.

5 Metodologia experimental

5.1 Clúster de computació

Situat a la Universitat de Rutgers, als Estat Units, és un clúster d'altes prestacions compost per vuit nodes. Aquest clúster disposa d'un Raritan com a instrumentació de mesura d'energia i un Yokogawa per mesurar el voltatge i la intensitat de cada node. Gràcies a aquests aparells, s'han pogut aconseguir totes les mesures de consum d'energia obtingudes en aquest treball.

Cada servidor té la següent configuració de maquinari:

- 2 x Intel Xeon E5-2690 v2 @ 3,00 Ghz. Aportant un total de 16 nuclis.
- 128 Gb DRAM
- Emmagatzematge:
    - 1 TB NVRAM (Fusion-IO IoDrive2)
    - 2 TB en discos d'estat sòlid (SSD)
    - 4 TB en discos durs tradicionals (HDD)
- Coprocessadors Intel Xeon Phi 7120P. Amb 16 GB de RAM, 61 nuclis i 244 fils d'execució.
- Xarxa:
    - Infiniband que proporciona 56 Gbps
    - Ethenet a 10 Gigabit.

Podem trobar les característiques completes al següent enllaç.

5.2 Metodologia dels càlculs

5.2.1 Programa de multiplicació de matrius

L'energia gastada en el procés de càlcul es mesura mitjançant un comptador d'energia ja instal·lat i disponible als clúster de computació. Aquest programa monitoritza les PDUs (Unitats de Distribució d'Energia) i extreu els consums de cada servidor del clúster utilitzat pel programa. Mitjançant un simple script, es poden extreure els resultats de consum d'energia de cada servidor a un fitxer de text des d'on podrem extreure els resultats per revisar, sumar-los finalment i obtenir les xifres d'energia.
El programa el llancem mitjançant la comanda de Linux *time*, que ens retornarà el temps d'execució del programa.
Per poder automatitzar les execucions, donat que un experiment complet pot allargar-se més d'una hora, es creen scripts que iteren per tots els valors dels paràmetres d'entrada que considero adients en cada moment.

Per la versió síncrona de la suma de matrius, faig servir els següents valors com a paràmetres d'entrada:

- Mida de les matrius quadrades: SIZE = 100
- Nombre d'iteracions per cada conjunt de càlculs: ITERS = 60
- Desequilibri entre nombre de multiplicacions de matrius per fil d'execució: IMBALANCE = (0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100)

Realitzo 10 experiments diferents, 7 dels quals són complets i dels altres 3 falten algunes combinacions de valors de paràmetres d'entrada. Això va ser causat per coincidir amb d'altres experiments que s'estaven realitzant al clúster i com és lògic tenien més prioritat que les meves mesures. Un inconvenient que va impedir que en algunes ocasions no acabessin els meus experiments satisfactòriament.
No he vist la necessitat de completar aquestes repeticions donada la consistència dels resultats obtinguts en les sèries completes d'experiments.

En total obtinc 8 hores i 23 minuts de temps d'execució repartit entre les diferents combinacions dels paràmetres d'entrada.

Per la versió asíncrona de la multiplicació de matrius, mantinc els mateixos valors dels tres paràmetres d'entrada fets servir anteriorment. D'aquesta manera puc comparar fàcilment execucions fetes amb ambdós programes. A més a més, afegeixo el paràmetre de temps d'espera:

- Temps d'espera (en ns) del fil d'execució abans de preguntar per l'estat de la resta de fils: tv_nsec = (3000, 5000, 7000, 9000, 11000, 13000, 15000)

En aquest cas realitzo 6 experiments: tres complets i tres amb alguns valors incomplerts pels motius exposats de manca d'exclusivitat d'ús del clúster.

En total aconsegueixo 33 hores i 18 minuts de temps d'execució repartit entre les diferents combinacions dels paràmetres d'entrada.

5.2.2 Programa de càlcul del número π

La metodologia feta servir en aquest programa és igual per tots els programes. Extraiem el consum registrat per les PDUs mitjançant software i calculem el temps d'execució llançant el programa amb la comanda *time* de Linux.
Llenço scripts d'execució per poder automatitzar la recollida de dades.

El paràmetre utilitzat en el programa síncron és:
- npts, nombre de rectangles generats: 10.000.000.000

Pel programa asíncron hem de tenir en compte també el temps d'espera i el desequilibri:
- tv_nsec, amb valors: 3000, 5000, 7000, 9000, 11000, 13000, 15000
- IMBALANCE: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

Com que per aquests càlculs vaig poder tenir en exclusivitat el clúster de computació, vaig realitzar una primera sèrie de 20 experiments síncrons. Veient que els resultats eren equilibrats i conseqüents, no vaig trobar la necessitat de fer més experiments. Ja disposava d'unes bones mesures de referència per poder comparar els resultats asíncrons.

Pel programa asíncron sí que realitzo repeticions dels experiments. En total en faig 11. Tots complets. En total són una mica més de 6 hores de computació.

6 Resultats

6.1 Programa de multiplicació de matrius

Fent la mitjana dels resultats de l'execució síncrona, obtenim els valors de referència amb que compararem els resultats obtinguts de manera asíncrona.

L'esmentada falta d'exclusivitat en l'ús del clúster fa que determinades mesures coincideixin amb altres treballs que s'executen en aquell moment, fent augmentar els valors de consum d'energia i de temps d'execució. Com que aquests valors són clarament visibles per la seva llunyania amb els obtinguts en d'altres experiments, fan que siguin fàcilment eliminats sense necessitat de fer servir cap mètode estadístic. Val a dir, que per la diferència horària existent entre els Estats Units i Catalunya, aquests problemes han estat mínims, ja que els meus horaris de treball al clúster solien coincidir amb les hores de mínima activitat d'allà.

Obtenim dos tipus de resultats: energia consumida i temps d'execució. A part de les taules amb els resultats ordenats, confecciono diversos tipus de gràfiques que m'ajuden a veure com es comporten les execucions dels diferents experiments.
En totes aquestes gràfiques, es mostren els valors asíncrons en línia continua i els valors síncrons (de referència) en línia discontínua.

Les taules creades són:

- Consum energètic vs. IMBALANCE (Figura 1), on es pot veure com influeix el desequilibri de càlculs entre fils d'execució en el consum energètic. A mida que augmentem el desequilibri, augmenta el consum d'energia. Excepte per valors de desequilibri igual a zero, la resta de valors queden per sobre del valor de referència síncron. Per tant, només millorem consum si no afegim desequilibri als fils d'execució.
- Temps d'execució vs. IMBALANCE (Figura 3), on tenim el temps d'execució dels programes en funció del grau de desequilibri introduit. Fins als valors d'IMBALANCE = 60, es pot veure que el temps d'execució és gairebé equivalent entre el programa síncron i l'asíncron. És més, els valors asíncrons sense desequilibri són millors que els trobats amb crides síncrones. Un cop més, es pot comporvar que alts nivells de desequilibri fan augmentar el temps d'execució ostensiblement.
- Consum energètic vs. Temps d'espera del fil d'execució (tv_nseq) (Figura 2), on podem observar el consum d'energia segons el grau de desequilibri introduït als càlculs del fil i segons el temps d'espera introduït al programa asíncron. Tornem a veure que valors grans de desequilibri fan augmentar el consum d'energia. Tots es valors queden per sobre de la seva referència síncrona excepte pels valors amb IMBALANCE = 0.
- Temps d'execució vs. Temps d'espera del fil d'execució (tv_nseq) (Figura 4). Aqui tenim representats els temps d'execució per temps d'espera i per sèries de desequilibri. Tormen a apreciar alts temps d'execució per aquells valors alts de desequilibri. Els millors valors de temps els trobem als experiments asíncrons amb desequilibri nul, que a més, queden per sota dels valors de referència síncrons.
- Consum energètic + temps d'execució vs. IMBALANCE per cada temps d'espera del fil d'execució fet servir. (Figures 5, 6, 7, 8, 9, 10 i 11). Aqui he representat per

separat cada temps d'espera fet servir en els experiments. Així, per cada valor de desequilibri tenim el temps d'execució i el consum energètic. La tònica general és que un augment del desequilibri augmenta el consum i el temps d'execució. I que els millors resultats els trobem allà on el desequilibri és igual a zero. En aquests casos, també s'arriba a millorar el valor de referència que és el temps d'execució del programa amb barreres col·lectives síncrones.

6.2 Programa de càlcul del número π

Aquesta vegada la comparació de resultats no es fa exactament igual que amb el programa de multiplicació de matrius, ja que he aplicat una part de les conclusions obtingudes del programa anterior. Veient que el desequilibri entre el grau de càlcul entre fils fa augmentar el temps d'execució en tots els casos, decideixo anar una mica més enllà i fer les comparacions de tots els resultats asíncrons únicament amb el millor resultat que es pot obtenir de manera síncrona, que és amb IMBALANCE = 0. És per això que no introdueixo el desequilibri en el càlcul síncron.

Una vegada ordenats en taules els resultats asíncrons, i donat que he realitzat 11 repeticions, elimino també aquells valors que veig que se surten de la tendència general. Però no només elimino els valors afectats sinó tots els obtinguts en aquell experiment concret. D'aquesta manera m'asseguro d'obtenir els millors resultats asíncrons possibles ja que he de tenir en compte que:

- El temps d'execució és molt curt i, per tant, les diferències entre els temps obtinguts no seran molt grans.
- Pel consum energètic passarà el mateix, ja que un temps petit d'execució implicarà un consum d'energia petit, minimitzant les diferències.

Realitzo també diverses gràfiques:

- Consum energètic vs. IMBALANCE (Figura 12), on tenim el consum d'energia en funció del desequilibri imposat. Hem de tenir en compte que el valor genèric de referència no fa servir cap desequilibri en els seus càlculs. Podem observar que els valors asíncrons sense desequilibri tenen millors temps d'execució que els de referència.
- Temps d'execució vs. IMBALANCE (Figura 14). Tenim representat el temps d'execució en funció del desequilibri i el temps d'espera. Es pot observar un augment gradual del temps d'execució amb el desequilibri. El que no es pot assegurar es que un augment del temps d'espera augmenti sempre el temps d'execució. Es pot apreciar que els valors amb desequilibri igual a zero, són els que tenen un millor resultat, millorant la referència.
- Consum energètic vs. Temps d'espera del fil d'execució (tv_nseq) (Figura 13). Aquí tornem a tenir el consum d'energia de l'aplicació en funció del temps d'espera i del grau de desequilibri. Es pot veure clarament que la sèria amb desequilibri nul, és la que millor resultats obté, bastant per sota del valor de referència.
- Temps d'execució vs. Temps d'espera del fil d'execució (tv_nseq) (Figura 15). Hem representat el temps d'execució de l'aplicació en funció del temps d'espera i amb sèries de dades per diferents percentatges de desequilibri. Valors grans de desequilibri tenen valors grans de temps d'execució. Un altre cop, la sèrie sense desequilibri obté els millors resultats.

- Consum energètic + temps d'execució vs. IMBALANCE per cada temps d'espera del fil d'execució fet servir. (Figures 16, 17, 18, 19, 20, 21 i 22). La representació separada per cada temps d'espera fa més clars els resultats obtinguts. És obvi que l'augment de desequilibri fa augmentar el temps d'execució i el consum d'energia. En tots els casos, el desequilibri nul obté els millors resultats i millora els valors de referència.

7 Conclusions

7.1 Programa de multiplicació de matrius

En tots els casos on el desequilibri de càlculs entre fils d'execució no existeix, el mètode asíncron és més eficient en termes de consum energètic.
En el cas del temps d'execució, obtenim una millora gairebé general en tots els casos plantejats. Com en el cas del consum energètic, quan no existeix desequilibri de càlculs entre fils, la millora es sempre efectiva. En el cas d'augmentar el desequilibri, fins a un valor d'IMBALANCE de 50 obtenim valors gairebé equivalents als síncrons per tots els temps d'espera. Només en els temps d'espera més alts, també obtenim igualtat de resultats amb els desequilibris més alts (90).

Les conclusions que podem extreure són les següents:

Si busquem millorar consum energètic, hem de distribuir equitativament els càlculs a realitzat entre tots els fils d'execució existents. L'estalvi d'energia ve donat perquè estem evitant sobrecarregar els processos en la consulta de l'estat de la resta de fils d'execució del programa. Com que estem reduint el nombre de consultes i a més, estem posant en espera els processos durant una part important de l'execució, és fàcil de trobar la millora del codi asíncron. Els percentatges de millora obtinguts van de l'11 al 22 %, sent millors els resultats obtinguts amb valors alts de temps d'espera del procés.

En canvi si busquem millorar temps d'execució, hem de continuar buscant l'equilibri de càlcul entre els fils d'execució. El percentatge de millora respecte als valors síncrons és situa al voltant del 32 %. Si augmentem el desequilibri entre processos, no obtenim cap millora en els resultats. En aquest cas, l'augment del temps d'espera no s'aprecia entre els diferents temps d'espera donat que són valors molt petits: De l'ordre dels 17-18 segons. Els percentatges de millora segons el temps d'espera van del 28% al 32%.
Però aquests valors no els considero fiables per la curta durada de l'experiment. Trobem millora entre els experiments asíncrons respecte els síncrons, però no tinc prou dades per assegurar com afecta en aquest cas el temps d'espera del procés.

Si mirem els experiments amb més durada, sí que podem observar una millora del temps d'execució en augmentar el temps d'espera. De manera equivalent al consum energètic, podem assegurar que amb fils d'execució amb càrrega de treball equivalent i temps d'espera grans, obtenim millora de temps d'execució respecte el mètode síncron.

Aquest guany en el temps d'execució el podem atribuir a la millor optimització de la comanda MPI_Test (feta servir en el procés asíncron per conèixer l'estat del càlcul pels diferents fils) respecte l'MPI_Barrier del procés síncron. MPI_Barrier és una crida bloquejant, mentre que MPI_Test no ho és pas. Això la fa més efectiva i el seu ús

augmenta el rendiment dels programes ja que actua de manera diferent a MPI_Barrier per aconseguir saber l'estat de la resta del fils d'execució del programa.

7.2 Programa de càlcul del número π

Tal i com es podia preveure, els millors resultats asíncrons en terme de temps d'execució i consum energètic els trobem en tots aquells experiments que no introdueixen desequilibri entre els fils d'execució.

El percentatge de millora en el temps d'execució és d'un 7% aproximadament.
Pel consum energètic, els guanys van del 2,95% al 7,57%, depenent del temps d'espera fet servir.
En aquest cas, els temps d'espera alts milloren més el consum energètic. Sent el valor òptim 7000 ns. Es pot observar també, que el menor temps d'espera és el que obté de llarg el pitjor resultat. Malgrat tot també millora el valor síncron.

Podem veure que la nova crida asíncrona MPI_Ireduce, torna a tenir un millor rendiment que la seva equivalent síncrona.
També hem de tenir en compte un detall: el càlcul del desequilibri afegeix una càrrega de càlcul al programa asíncron que no té el síncron. A més, aquest càlcul de desequilibri, incorpora una barrera síncrona. Per tant, el programa asíncron ni ho és en la seva totalitat, ni tampoc és del tot equivalent amb el seu rival síncron. Tot i això, aconsegueix millorar els resultats computacionals, donant una idea de la efectivitat de les noves crides asíncrones.

7.3 Conclussions generals

Com s'ha pogut veure, les crides asíncrones tenen un rendiment molt superior a les equivalents síncrones. Aquesta nova perspectiva en la programació, fa millorar encara més els resultats obtinguts amb els programes que fan servir MPI.
La computació d'altes prestacions ha permès assolir grans èxits en la investigació mundial i el fet de millorar una llibreria tan necessària pels científics, farà possible assolir noves fites, impensables d'aconseguir a un nivell tan "amateur" mirant uns anys enrere.

També s'ha pogut demostrar que desequilibrar els fils d'execució fa malbaratar recursos, ja que tenim uns processos fent càlculs mentre uns altres els tenim esperant una bona estona. Això fa que s'allarguin els temps d'execució i el consum energètic.
En igualtat de condicions (desequilibri entre fils) es pot observar que el sobrecost que incorporen les barreres síncrones és molt alt. Les noves crides asíncrones retornen més ràpidament i amb menys recursos l'estat dels fils quan es crida a MPI_Test.

La recomanació és ràpida: evitar les barreres síncrones. En la gran majoria dels casos, la transformació és trivial, però degut a que els programes d'investigació no són tan simples com aquests que he fet servir de test, es veu necessari anar una mica més enllà per realitzar aquestes millores de manera transparent. Ho explico en el següent apartat.

8 Treball futur

A continuació detallo els treballs que seguirien a aquest treball de final de màster, però que la falta de temps m'ha impedit desenvolupar.

8.1 Optimització/ampliació dels programes de proves

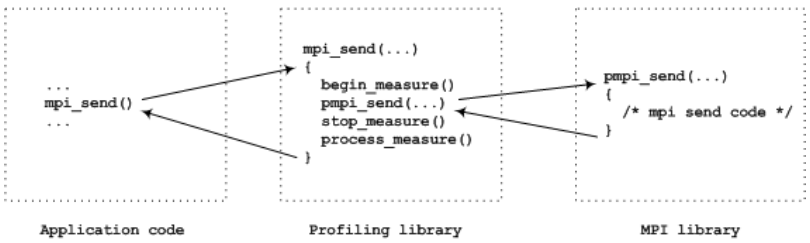
Una primera millora seria transformar el programa asíncron del número π: treure la barrera síncrona MPI_Bcast per la seva equivalent asíncrona MPI_Ibcast. D'aquesta manera reduiríem els sobrecost de càlcul i comunicacions que genera el repartiment del nombre de rectangles a calcular entre tots els fils d'execució i millorarìem encara més els temps d'execució i consum energètic del programa.

8.2 Substitució en temps d'execució de les crides síncrones

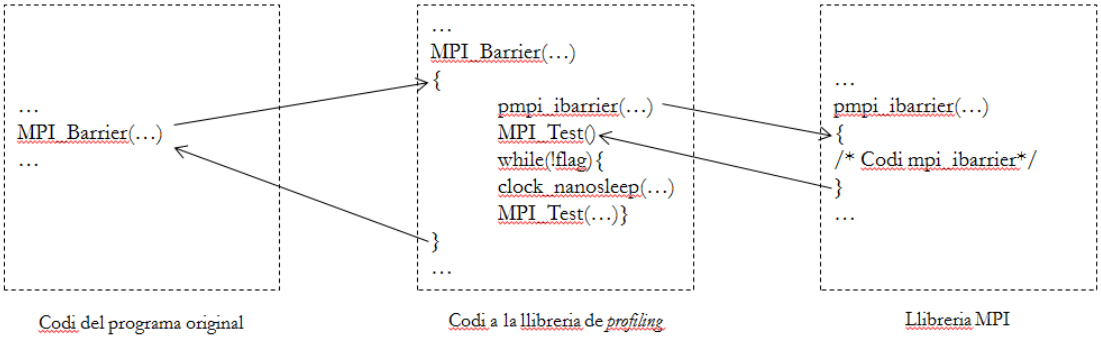
Aquesta seria l'actuació que tindria una aplicació més pràctica en l'àmbit d'us dels programes MPI.
El que es pretén es substituir les crides síncrones per les seves asíncrones de manera totalment transparent per l'usuari i en temps d'execució. És a dir, sense haver de tocar el codi original.
Això es pot realitzar amb les anomenades llibreries de *profiling* (pMPI).
Aquestes llibreries es fan servir per poder prendre mesures estadístiques de les funcions MPI d'un programa. Moltes presenten els resultats mitjançant gràfics permetent veure, per exemple, on triga més el programa a realitzar els càlculs i, amb aquesta informació, procedir a optimitzar aquella part de codi que resulta més ineficient.

Aquestes llibreries s'enllacen al programa MPI en temps d'execució de manera que, quan el programa original ha de cridar a una operació MPI, en comptes d'anar-la a buscar a la llibreria d'MPI, crida a la operació equivalent de pMPI. Aquesta operació equivalent és la que hem desenvolupat nosaltres i, normalment afegeix una marca de temps abans de cridar a la operació original que, aquesta vegada sí, es va a buscar a les llibreries d'MPI.

El seu funcionament es podria resumir de la següent manera:



Per tant, el que hauríem de fer és definir dintre de la llibreria de *profiling* com volem que es comporti el programa en el cas d'arribar a una determinada crida síncrona. El que voldrem és que en comptes de cridar al mètode síncron ho faci a l'asíncron. I un cop retorni aquest últim, veure les condicions de la resta de fils per començar l'espera o no. Seria un algoritme com el següent:



D'aquesta manera, aconseguim alliberar de l'optimització a l'usuari del programa, que potser no té els coneixement o el temps per anar modificant els programes que fa servir, i aconseguiríem ràpida i fàcilment una millora en els resultats obtinguts.

També s'hauria de donar la possibilitat a l'usuari de poder modificar els paràmetres que afecten directament al rendiment asíncron, com per exemple el temps d'espera. Jo seria partidari de posar uns paràmetres per defecte que s'han vist que funcionen amb els programes de test i fer servir els de l'usuari solament si els indiqués a l'hora de crida el programa.

8.3 Optimització automàtica dels paràmetres asíncrons

Una segona fase en l'optimització de programes MPI, seria la de que el programa sabés escollir aquells paràmetres que aportarien uns millors resultats. Alliberaríem totalment a l'usuari dels sobrecostos del funcionament d'MPI i podria centrar-se en l'optimització pura i dura dels càlculs aritmètics pròpiament dits.

9 Taules de resultats

9.1 Multiplicació de matrius

9.1.1 Experiments síncrons

IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	4195	15	5158	16	13264	102	4228	15
10	8050	67	7256	67	3149	4	8354	67
20	12701	118	12632	117			13149	118
30	18086	169	18290	169			18923	168
40	23732	221	23758	220			24718	221
50	29214	271	29231	271			30498	272
60	34721	322	35633	322			36144	322
70	60907	429	42222	374			48911	416
80			47612	430			83942	596
90							111909	773
100							131982	949

IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	4545	18	4330	16	4340	17	4332	16
10	7434	68	7348	66	8298	67	8199	66
20	13241	119	13135	118	13211	118	13195	119
30	18912	168	18639	166	19016	170	18842	168
40	24813	221	24640	220	24643	220	24747	221
50	30413	271	30382	271	30465	271	30470	272
60	36089	322	36066	322	36069	322	36061	322
70	49564	416	48162	418	48759	417	49695	416
80	87325	595	88959	593	95171	592	84235	590
90	118068	771	116570	775	120023	773	105378	779
100	126367	946	147284	950			131592	954

IMBALANCE	Consum	Temps	Consum	Temps		Consum Mitjà	Temps Mitjà
0	4432	18	4199	17		5302	25
10	15632	66	15640	66		8936	60
20	27542	118	27557	117		16263	118
30	39451	168	39464	169		23291	168
40	51350	220	51136	219		30393	220
50	63254	271	63366	270		37477	271
60	74913	320	75038	321		44526	322
70	97662	420	96966	416		60316	414
80	137493	592	137766	591		95313	572
90	179598	774	189706	818		134465	780
100	221406	949	222537	957		163528	951

9.1.2 Experiments asíncrons

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
3000	**0**	4006	16	4436	16	4297	18
3000	**10**	16010	67	15642	67	6959	66
3000	**20**	28029	118	27568	118	21119	117
3000	**30**	39791	169	39189	168	29676	168
3000	**40**	52060	221	51099	221	44908	218
3000	**50**	63862	271	63015	270	60630	270
3000	**60**	76166	322	75158	322	72993	322
3000	**70**	112755	477	99451	475	211210	427
3000	**80**	155979	661	138896	660	391293	595
3000	**90**	199715	847	179780	845	563643	772
3000	**100**	243424	1033	220625	1153	861350	950

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
3000	0	3715	15	4829	18	4430	18
3000	10	15572	66	17279	66	15391	65
3000	20	27404	117	30178	117	27535	117
3000	30	-	-	43098	167	39219	168
3000	40	-	-	56247	219	51137	218
3000	50	62704	270	69492	271	62979	270
3000	60	74500	321	82169	321	74893	321
3000	70	96815	417	104052	415	97529	418
3000	80	136983	591	138721	596	137645	591
3000	90	179230	773	181161	778	180370	775
3000	100	219407	947	220805	949	251601	1080

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
5000	0	4007	16	5296	16	4437	18
5000	10	16021	68	6960	67	15662	66
5000	20	28009	118	22888	118	27335	117
5000	30	-	-	33344	169	39290	168
5000	40	52044	220	46554	220	51235	219
5000	50	64273	272	59820	270	63049	270
5000	60	75355	319	71822	320	74984	320
5000	70	111758	474	209457	475	97151	417
5000	80	398987	655	508351	792	138225	593
5000	90	529752	847	568074	850	181000	776
5000	100	708967	1030	749980	1051	220736	951

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
5000	**0**	-	-	4429	18	4441	18
5000	**10**	-	-	15620	66	15402	65
5000	**20**	37607	117	27512	117	27531	118
5000	**30**	54480	167	39401	169	39197	167
5000	**40**	65822	220	51056	219	51138	219
5000	**50**	66095	271	62949	269	63250	271
5000	**60**	86335	321	74837	321	74964	321
5000	**70**	110935	418	97455	419	97313	417
5000	**80**	152297	593	137790	591	138915	596
5000	**90**	200716	773	181406	779	179269	769
5000	**100**	235857	952	221379	951	221483	951

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
7000	**0**	5301	16	5256	16	4420	18
7000	**10**	7779	66	7831	67	15566	66
7000	**20**	13100	119	27327	119	27438	118
7000	**30**	27179	168	41331	169	39297	168
7000	**40**	34171	220	54603	221	50895	219
7000	**50**	-	-	66042	272	62782	270
7000	**60**	67397	319	78799	319	74633	321
7000	**70**	199812	473	220357	473	96959	417
7000	**80**	373392	660	397786	664	136927	590
7000	**90**	554982	846	582105	848	178998	771
7000	**100**	723111	1036	757253	1039	220445	950

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
7000	0	4413	18	4432	18	4429	18
7000	10	15324	66	15619	66	15630	66
7000	20	27213	116	27284	116	27554	117
7000	30	39281	169	39402	169	39227	167
7000	40	50857	218	51293	220	51321	220
7000	50	62722	270	63186	271	63306	270
7000	60	74509	321	74837	321	75194	322
7000	70	97319	419	97448	418	97480	417
7000	80	137242	592	171838	738	138259	593
7000	90	178962	773	180227	774	180767	775
7000	100	219833	949	221476	952	222328	954

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
9000	0	3484	16	5284	16	4415	18
9000	10	6877	66	6842	66	15567	66
9000	20	16671	118	22761	117	27422	118
9000	30	28052	169	39527	169	39060	167
9000	40	37728	220	52699	219	51177	220
9000	50	56388	271	54438	272	62768	270
9000	60	67379	320	81789	322	74616	321
9000	70	202398	472	221574	476	97388	419
9000	80	-	-	403832	658	138339	596
9000	90	553381	845	582857	852	179252	772
9000	100	718180	1033	758292	1034	221129	953

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
9000	0	4408	19	4427	18	4434	18
9000	10	15548	66	15388	65	15664	66
9000	20	27382	117	27509	118	27552	117
9000	30	39219	169	39403	168	39187	168
9000	40	51097	219	51056	219	51379	220
9000	50	62683	270	62960	270	63255	270
9000	60	74493	321	74853	321	74937	321
9000	70	96578	416	97463	418	96882	416
9000	80	137213	592	138094	593	138693	594
9000	90	179279	774	-	-	180487	775
9000	100	220291	950	222653	953	222567	955

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
11000	0	3491	16	-	18	3722	15
11000	10	6967	68	6955	68	15354	65
11000	20	19338	118	19294	118	27440	118
11000	30	32583	170	42214	168	39098	167
11000	40	43894	219	56410	221	51142	220
11000	50	54391	271	74965	272	63037	271
11000	60	68547	322	77311	322	74676	321
11000	70	-	-	216307	477	96722	416
11000	80	-	-	-	-	137400	592
11000	90	-	-	-	-	180464	777
11000	100	-	-	-	-	219756	947

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
11000	0	4412	18	4448	18	4441	18
11000	10	15315	65	15684	66	15636	66
11000	20	27394	117	27627	117	27565	117
11000	30	39071	168	39559	169	39459	169
11000	40	51219	219	51273	219	51151	219
11000	50	62951	270	63212	269	63025	269
11000	60	74858	321	75144	321	74938	321
11000	70	96772	415	97277	418	97372	418
11000	80	138025	593	137794	591	138065	591
11000	90	179045	770	179886	773	180096	773
11000	100	221243	953	222686	948	220944	948

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
13000	0	4403	16	-	-	3715	15
13000	10	7851	67	-	-	15587	67
13000	20	18452	119	-	-	27449	117
13000	30	29785	168	-	-	39270	168
13000	40	39547	221	-	-	50943	219
13000	50	53757	272	-	-	62787	270
13000	60	69394	322	-	-	74911	322
13000	70	-	-	-	-	97005	417
13000	80	-	-	-	-	137949	594
13000	90	-	-	-	-	180222	777
13000	100	-	-	-	-	221157	954

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
13000	0	4408	18	4461	18	4433	18
13000	10	15599	66	22577	66	15642	66
13000	20	27149	116	42246	117	27317	116
13000	30	38983	168	60821	168	39470	169
13000	40	51044	219	55787	219	51391	219
13000	50	62687	270	63189	270	63289	271
13000	60	74525	321	80085	321	74972	321
13000	70	97746	419	97872	418	97629	419
13000	80	138342	594	-	-	138260	592
13000	90	180182	775	180554	772	180128	773
13000	100	221733	953	220717	948	-	-

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
15000	0	-	-	-	-	3487	14
15000	10	-	-	-	-	15336	66
15000	20	-	-	-	-	27209	116
15000	30	-	-	-	-	39278	169
15000	40	-	-	-	-	50931	218
15000	50	-	-	-	-	63038	271
15000	60	-	-	-	-	74642	321
15000	70	-	-	-	-	97177	418
15000	80	-	-	-	-	137384	592
15000	90	-	-	-	-	179121	773
15000	100	-	-	-	-	-	-

Temps d'espera	IMBALANCE	Consum	Temps	Consum	Temps	Consum	Temps
15000	0	4427	18	4427	19	-	-
15000	10	15613	66	15380	65	-	-
15000	20	27496	117	27505	117	-	-
15000	30	39080	168	39159	168	-	-
15000	40	51086	219	51282	219	-	-
15000	50	62892	271	63169	271	-	-
15000	60	74803	322	75064	322	-	-
15000	70	97097	418	97438	418	-	-
15000	80	-	-	138233	594	-	-
15000	90	179147	773	179956	773	-	-
15000	100	220466	951	222371	956	-	-

Temps d'espera	IMBALANCE	Consum Mitjà	Temps Mitjà
3000	0	4285,5	17
3000	10	14475,5	66
3000	20	26972,167	117
3000	30	38194,6	168
3000	40	51090,2	219
3000	50	63780,333	270
3000	60	75979,833	322
3000	70	120302	438
3000	80	183252,83	616
3000	90	247316,5	798
3000	100	336202	1019

Temps d'espera	IMBALANCE	Consum Mitjà	Temps Mitjà
5000	0	4522	17
5000	10	13933	66
5000	20	28480,333	118
5000	30	41142,4	168
5000	40	52974,833	220
5000	50	63239,333	271
5000	60	76382,833	320
5000	70	120678,17	437
5000	80	245760,83	637
5000	90	306702,83	799
5000	100	393067	981

Temps d'espera	IMBALANCE	Consum Mitjà	Temps Mitjà
7000	**0**	4708,5	17
7000	10	12958,167	66
7000	**20**	24986	118
7000	30	37619,5	168
7000	**40**	48856,667	220
7000	50	63607,6	271
7000	**60**	74228,167	321
7000	70	134895,83	436
7000	**80**	225907,33	640
7000	90	309340,17	798
7000	**100**	394074,33	980

Temps d'espera	IMBALANCE	Consum Mitjà	Temps Mitjà
9000	0	4408,6667	18
9000	10	12647,667	66
9000	20	24882,833	118
9000	30	37408	168
9000	40	49189,333	220
9000	50	60415,333	271
9000	60	74677,833	321
9000	70	135380,5	436
9000	80	191234,2	607
9000	90	335051,2	804
9000	100	393852	980

Temps d'espera	IMBALANCE	Consum Mitjà	Temps Mitjà
11000	0	4102,8	17
11000	10	12651,833	66
11000	20	24776,333	118
11000	30	38664	169
11000	40	50848,167	220
11000	50	63596,833	270
11000	60	74245,667	321
11000	70	120890	429
11000	80	137821	592
11000	90	179872,75	773
11000	100	221157,25	949

Temps d'espera	IMBALANCE	Consum Mitjà	Temps Mitjà
13000	0	4284	17
13000	10	15451,2	66
13000	20	28522,6	117
13000	30	41665,8	168
13000	40	49742,4	219
13000	50	61141,8	271
13000	60	74777,4	321
13000	70	97563	418
13000	80	138183,67	593
13000	90	180271,5	774
13000	100	221202,33	952

Temps d'espera	IMBALANCE	Consum Mitjà	Temps Mitjà
15000	0	4113,6667	17
15000	10	15443	66
15000	20	27403,333	117
15000	30	39172,333	168
15000	40	51099,667	219
15000	50	63033	271
15000	60	74836,333	322
15000	70	97237,333	418
15000	80	137808,5	593
15000	90	179408	773
15000	100	221418,5	954

9.2 Càlcul del número π

9.2.1 Experiments síncrons

Experiment	Consum	Temps
1	3040	12
2	3691	14
3	3532	13
4	3539	13
5	3695	14
6	3537	13
7	3790	25
8	3535	13
9	3537	13
10	3529	13
11	3528	13
Mitjana	3541	14

9.2.2 Experiments asíncrons

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	3000	~~6090~~	~~25~~	3292	13	3760	15	3525	14
10	3000	~~3989~~	~~16~~	3995	17	4004	17	3998	16
20	3000	~~3981~~	~~16~~	3998	16	3995	16	4702	19
30	3000	~~5394~~	~~22~~	3995	16	4704	19	4472	18
40	3000	~~5384~~	~~22~~	5409	22	6110	25	6112	25
50	3000	~~5153~~	~~21~~	6819	28	4700	19	6826	28
60	3000	~~7495~~	~~31~~	6825	28	8229	34	5414	22
70	3000	~~7492~~	~~32~~	8939	37	9646	40	6116	25
80	3000	~~7495~~	~~31~~	9645	40	6815	28	9648	41
90	3000	~~8916~~	~~37~~	9639	40	8001	33	7998	33
100	3000	~~11507~~	~~48~~	7520	32	10348	43	8937	37

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	3000	3295	13	3296	13	3766	15	3307	14
10	3000	3996	16	3997	16	4009	16	4025	16
20	3000	4701	19	4707	19	5419	22	4725	19
30	3000	4702	19	4702	19	6127	25	6147	25
40	3000	5410	22	8237	34	4712	19	4732	19
50	3000	5407	22	4703	19	5421	23	6156	25
60	3000	6827	28	5410	22	5415	22	5437	22
70	3000	5405	22	7524	31	5417	22	5208	21
80	3000	9644	40	6815	28	6130	25	8275	34
90	3000	11771	49	8943	37	8947	37	9692	40
100	3000	7520	31	10827	45	8982	37	11828	49

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum Mitjà	Temps Mitjà
0	3000	3067	12	3769	15	3290	13	3437	14
10	3000	4007	16	4005	16	3293	13	3933	16
20	3000	6832	28	4001	16	3997	16	4708	19
30	3000	4709	20	5413	22	4712	19	4968	20
40	3000	7533	31	4706	19	6118	25	5908	24
50	3000	6838	28	5415	22	8003	33	6029	25
60	3000	6833	28	6124	25	4709	19	6122	25
70	3000	7540	31	6123	25	7535	31	6945	29
80	3000	6843	28	6828	28	6824	28	7747	32
90	3000	9662	40	6828	28	6828	28	8831	37
100	3000	11786	49	11775	50	8240	34	9776	41

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	5000	3284	13	3295	13	3296	13	3290	13
10	5000	3980	16	3995	16	3995	17	4000	16
20	5000	3979	16	4001	17	4704	19	3996	16
30	5000	4685	19	3995	17	4700	19	4707	19
40	5000	5383	22	4708	19	5410	22	5171	21
50	5000	6090	25	5172	21	5405	22	8234	34
60	5000	6100	25	6113	25	6821	28	7524	31
70	5000	6101	26	7520	31	8937	37	8225	34
80	5000	8199	34	8226	34	7520	31	8225	35
90	5000	11003	46	9641	40	8933	37	7524	31
100	5000	9597	40	7998	33	7523	31	11759	49

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	5000	3294	13	3305	14	3311	13	3309	13
10	5000	3995	17	3997	16	4017	16	4026	17
20	5000	3995	16	3997	16	4723	19	4728	19
30	5000	3995	16	4700	19	5198	21	6145	25
40	5000	4700	19	4700	19	4735	19	6145	25
50	5000	5413	22	5405	22	6857	28	6158	25
60	5000	6118	25	4705	19	6153	25	5447	22
70	5000	6110	25	6815	28	6626	28	8038	33
80	5000	11757	49	5411	22	6849	28	7570	31
90	5000	9635	40	11523	48	6850	28	8989	37
100	5000	11764	49	8936	37	9696	40	6154	25

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum Mitjà	Temps Mitjà
0	5000	3298	13	~~6120~~	~~25~~	3294	13	3298	13
10	5000	3298	13	~~4005~~	~~16~~	4006	17	3931	16
20	5000	6125	26	~~4001~~	~~16~~	4001	16	4425	18
30	5000	4715	20	~~5415~~	~~22~~	4470	18	4731	19
40	5000	5430	22	~~5884~~	~~24~~	5420	22	5180	21
50	5000	4476	18	~~6125~~	~~25~~	6122	25	5933	24
60	5000	5423	22	~~5418~~	~~22~~	7529	31	6193	25
70	5000	6833	28	~~8249~~	~~34~~	6823	28	7203	30
80	5000	8245	34	~~8958~~	~~37~~	6829	28	7883	33
90	5000	11790	49	~~6125~~	~~25~~	8238	34	9413	39
100	5000	10366	43	~~8953~~	~~38~~	8951	37	9274	38

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	7000	3280	13	3291	13	3298	13	3301	13
10	7000	3981	16	3999	16	3995	16	3997	16
20	7000	4686	19	3996	16	3995	17	4001	16
30	7000	5401	22	4712	19	5415	22	6112	25
40	7000	5389	22	5406	23	7298	30	5413	22
50	7000	6788	28	5413	22	6110	25	5405	22
60	7000	7951	27	6821	28	6824	28	7527	31
70	7000	9131	30	6816	28	7525	31	5409	22
80	7000	10039	33	8231	34	8229	34	8232	35
90	7000	11521	37	10348	43	8225	34	9643	40
100	7000	13405	45	11052	46	11059	46	9639	40

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	7000	3290	14	3290	13	3311	13	3309	13
10	7000	3995	16	3291	13	4015	16	4025	16
20	7000	4007	16	3996	17	4726	19	4018	17
30	7000	4700	19	4701	19	4726	19	5677	23
40	7000	5415	22	7524	31	4728	19	6149	25
50	7000	6117	25	7529	31	6138	25	4731	19
60	7000	6111	25	7528	31	6853	28	7569	31
70	7000	7290	30	8238	34	7565	31	5444	22
80	7000	7526	31	8234	34	8269	35	8995	37
90	7000	8234	34	10348	43	7563	31	7568	31
100	7000	7520	31	8225	34	9692	40	11828	49

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum Mitjà	Temps Mitjà
0	7000	3303	13	3297	13	3298	13	3297	13
10	7000	6119	25	4003	16	4003	16	4129	17
20	7000	4712	19	3303	13	4005	16	4131	17
30	7000	4708	19	5181	21	4709	19	5095	21
40	7000	6124	25	5427	22	5417	22	5845	24
50	7000	4709	19	7541	31	6118	25	6054	25
60	7000	6128	25	6827	28	7531	31	7061	28
70	7000	6598	27	7305	30	6828	28	7104	28
80	7000	7701	31	8250	34	5413	22	8102	33
90	7000	10786	43	6135	25	7294	30	8879	36
100	7000	11232	46	6830	28	11535	48	10183	41

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	9000	3293	13	3294	13	3290	13	3290	13
10	9000	3981	16	6110	25	3995	16	3290	13
20	9000	3990	16	4705	19	4706	20	3999	16
30	9000	4687	19	4700	19	3995	16	4703	19
40	9000	9243	31	4704	20	6112	25	4709	19
50	9000	5956	18	5406	22	6110	25	4700	19
60	9000	7831	24	4710	19	6118	25	6824	27
70	9000	9729	30	7534	31	5415	22	6113	25
80	9000	9486	34	8225	34	10348	43	7533	31
90	9000	12165	43	6825	28	9416	39	8234	35
100	9000	15405	42	11047	46	8227	34	11053	46

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	9000	3297	13	3295	13	3308	13	~~6152~~	~~25~~
10	9000	3995	16	3996	16	4020	16	~~4028~~	~~16~~
20	9000	4704	19	4002	17	6152	25	~~4021~~	~~17~~
30	9000	3995	16	4702	19	5444	22	~~4738~~	~~19~~
40	9000	5405	22	6823	28	6151	25	~~6150~~	~~25~~
50	9000	5405	22	8229	34	6148	25	~~7576~~	~~31~~
60	9000	6122	25	4700	19	4735	19	~~5444~~	~~22~~
70	9000	6815	28	8931	37	8277	34	~~5436~~	~~22~~
80	9000	9643	40	6821	28	7563	32	~~10521~~	~~43~~
90	9000	10119	42	7527	31	10408	43	~~6183~~	~~25~~
100	9000	11053	46	11766	49	10873	45	~~6922~~	~~28~~

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum Mitjà	Temps Mitjà
0	9000	3295	13	3300	13	~~6124~~	~~25~~	3296	13
10	9000	4005	16	3999	17	~~4004~~	~~16~~	4155	17
20	9000	4240	17	4006	16	~~4705~~	~~19~~	4500	18
30	9000	4709	20	4003	16	~~6124~~	~~26~~	4549	18
40	9000	6900	28	4709	19	~~6118~~	~~25~~	6084	24
50	9000	5592	22	5174	21	~~6128~~	~~25~~	5858	23
60	9000	10780	43	6827	28	~~5413~~	~~22~~	6516	25
70	9000	7536	31	5414	22	~~5411~~	~~22~~	7307	29
80	9000	10358	43	10356	43	~~6818~~	~~28~~	8926	36
90	9000	8950	37	11774	49	~~7518~~	~~31~~	9491	39
100	9000	11774	49	7998	33	~~8933~~	~~37~~	11022	43

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	11000	~~5788~~	~~15~~	3295	13	~~6118~~	~~25~~	3295	13
10	11000	~~4089~~	~~15~~	3995	16	~~4000~~	~~16~~	3995	16
20	11000	~~6079~~	~~25~~	4703	19	~~3995~~	~~17~~	4701	19
30	11000	~~5387~~	~~22~~	5410	22	~~4700~~	~~19~~	4706	19
40	11000	~~5384~~	~~22~~	4707	19	~~6117~~	~~25~~	5407	22
50	11000	~~5158~~	~~21~~	6111	26	~~6119~~	~~25~~	6588	27
60	11000	~~11052~~	~~40~~	5413	22	~~6815~~	~~28~~	6148	25
70	11000	~~25189~~	~~70~~	6816	28	~~6119~~	~~25~~	8937	37
80	11000	~~22503~~	~~60~~	6580	27	~~8934~~	~~37~~	7521	31
90	11000	~~25917~~	~~69~~	8229	34	~~8226~~	~~34~~	6582	28
100	11000	~~34816~~	~~93~~	8940	37	~~6111~~	~~25~~	8945	37

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	11000	~~6110~~	~~25~~	3290	13	3310	13	3314	13
10	11000	~~3999~~	~~17~~	3995	16	4018	16	4020	16
20	11000	~~3995~~	~~16~~	4005	17	4731	19	4024	16
30	11000	~~5411~~	~~22~~	5406	23	5201	21	4492	18
40	11000	~~6816~~	~~28~~	4705	19	6148	25	5443	23
50	11000	~~5178~~	~~21~~	6114	25	4734	19	6149	25
60	11000	~~8701~~	~~36~~	12461	52	5910	24	8274	34
70	11000	~~6110~~	~~25~~	7527	31	7573	31	10171	42
80	11000	~~10343~~	~~43~~	7522	31	6862	29	6859	28
90	11000	~~6816~~	~~28~~	6117	25	10404	43	6160	25
100	11000	~~8940~~	~~37~~	11520	48	9696	40	8986	37

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum Mitjà	Temps Mitjà
0	11000	3298	13	3293	13	3294	13	3299	13
10	11000	3999	16	3998	16	3996	16	4002	16
20	11000	4008	16	5415	22	4003	16	4449	18
30	11000	4709	19	4704	19	5405	22	5004	20
40	11000	6126	25	5180	21	5411	23	5391	22
50	11000	6126	25	5415	22	5408	22	5831	24
60	11000	5414	22	5421	22	8228	34	7159	29
70	11000	6833	28	6828	28	5404	22	7511	31
80	11000	8240	34	8237	34	7520	31	7418	31
90	11000	8247	35	9645	40	7522	31	7863	33
100	11000	10368	43	7523	31	8934	37	9364	39

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	13000	~~10363~~	~~27~~	3290	13	3294	13	3290	13
10	13000	~~18137~~	~~49~~	4007	16	3995	16	4008	16
20	13000	~~13666~~	~~36~~	3997	16	4710	19	3998	16
30	13000	~~17021~~	~~45~~	6122	25	5405	23	4714	19
40	13000	~~16010~~	~~42~~	7531	31	4711	20	5406	22
50	13000	~~20613~~	~~54~~	5405	22	4700	19	6823	28
60	13000	~~25336~~	~~66~~	7525	32	6825	28	8936	37
70	13000	~~21962~~	~~58~~	5408	22	7524	31	7527	31
80	13000	~~21952~~	~~57~~	5409	22	7526	31	5405	22
90	13000	~~10047~~	~~39~~	9641	40	10348	43	-	-
100	13000	~~8235~~	~~34~~	9644	40	10352	43	-	-

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	13000	3290	13	3292	13	3075	12	3318	13
10	13000	4002	17	3995	16	4022	16	4018	16
20	13000	3995	16	3995	17	4018	16	4027	16
30	13000	5417	22	4710	19	4727	19	4492	18
40	13000	5406	22	5405	22	5446	22	4966	21
50	13000	5415	22	5417	22	5442	22	6616	27
60	13000	6116	25	7533	31	6149	25	5443	22
70	13000	9641	40	6111	25	6150	25	6861	28
80	13000	7285	30	6589	27	11828	50	6858	28
90	13000	7997	33	11757	49	9687	40	8990	37
100	13000	11055	46	11758	50	11824	49	11816	49

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum Mitjà	Temps Mitjà
0	13000	3301	13	3293	13	3289	13	3273	13
10	13000	4004	17	3761	15	4000	16	3981	16
20	13000	3770	15	4710	20	3994	16	4121	17
30	13000	4709	19	4706	19	4000	16	4900	20
40	13000	4715	19	4709	19	5404	22	5370	22
50	13000	5420	22	5409	22	6110	25	5676	23
60	13000	6831	28	5413	22	6107	26	6688	28
70	13000	8011	33	9643	40	11754	49	7863	32
80	13000	7539	31	8232	34	6813	28	7348	30
90	13000	9664	40	8230	34	9637	40	9550	40
100	13000	8244	34	7534	31	8931	37	10129	42

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	15000	3296	13	3290	13	3290	13	-	-
10	15000	10722	31	4000	16	3997	16	-	-
20	15000	13323	36	4701	19	4002	16	-	-
30	15000	14699	40	4701	19	5407	22	-	-
40	15000	14785	39	5407	22	5411	23	-	-
50	15000	24850	66	6120	25	6113	25	-	-
60	15000	21729	57	5877	24	6112	25	-	-
70	15000	20602	54	8238	34	8003	33	-	-
80	15000	29647	79	6827	28	7532	31	-	-
90	15000	34401	90	9648	40	11764	49	-	-
100	15000			7528	31	11525	48	-	-

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum	Temps
0	15000	3294	13	3296	13	~~6146~~	~~25~~	3309	13
10	15000	3995	16	3290	13	~~4017~~	~~16~~	4019	16
20	15000	4701	19	4708	19	~~5449~~	~~22~~	4019	16
30	15000	5405	22	4466	18	~~4019~~	~~16~~	4014	16
40	15000	5411	22	6116	25	~~4728~~	~~19~~	4728	19
50	15000	5405	22	6110	25	~~6146~~	~~25~~	5434	23
60	15000	5415	22	5408	22	~~4731~~	~~19~~	6148	25
70	15000	7533	31	10343	43	~~6862~~	~~28~~	7564	31
80	15000	7521	31	9648	40	~~8985~~	~~38~~	6140	25
90	15000	8238	34	6111	25	~~8278~~	~~34~~	11112	46
100	15000	9641	40	11771	49	~~8754~~	~~36~~	11828	49

IMBALANCE	Temps d'espera	Consum	Temps	Consum	Temps	Consum	Temps	Consum Mitjà	Temps Mitjà
0	15000	3297	13	3296	13	3293	13	3296	13
10	15000	4000	16	4006	16	3758	15	4643	17
20	15000	4715	20	4000	16	3992	16	5351	20
30	15000	5414	22	4710	19	4699	19	5946	22
40	15000	5416	22	4705	20	4699	19	6298	23
50	15000	5412	22	5415	22	6114	25	7886	28
60	15000	5416	22	6117	25	6115	26	7593	28
70	15000	8949	37	6821	28	8221	34	9586	36
80	15000	7534	31	10123	42	6820	28	10199	37
90	15000	8951	37	8945	37	6113	25	11698	43
100	15000	8954	37	11767	49	9634	40	10331	43

10 Gràfiques de resultats

10.1 Multiplicació de matrius

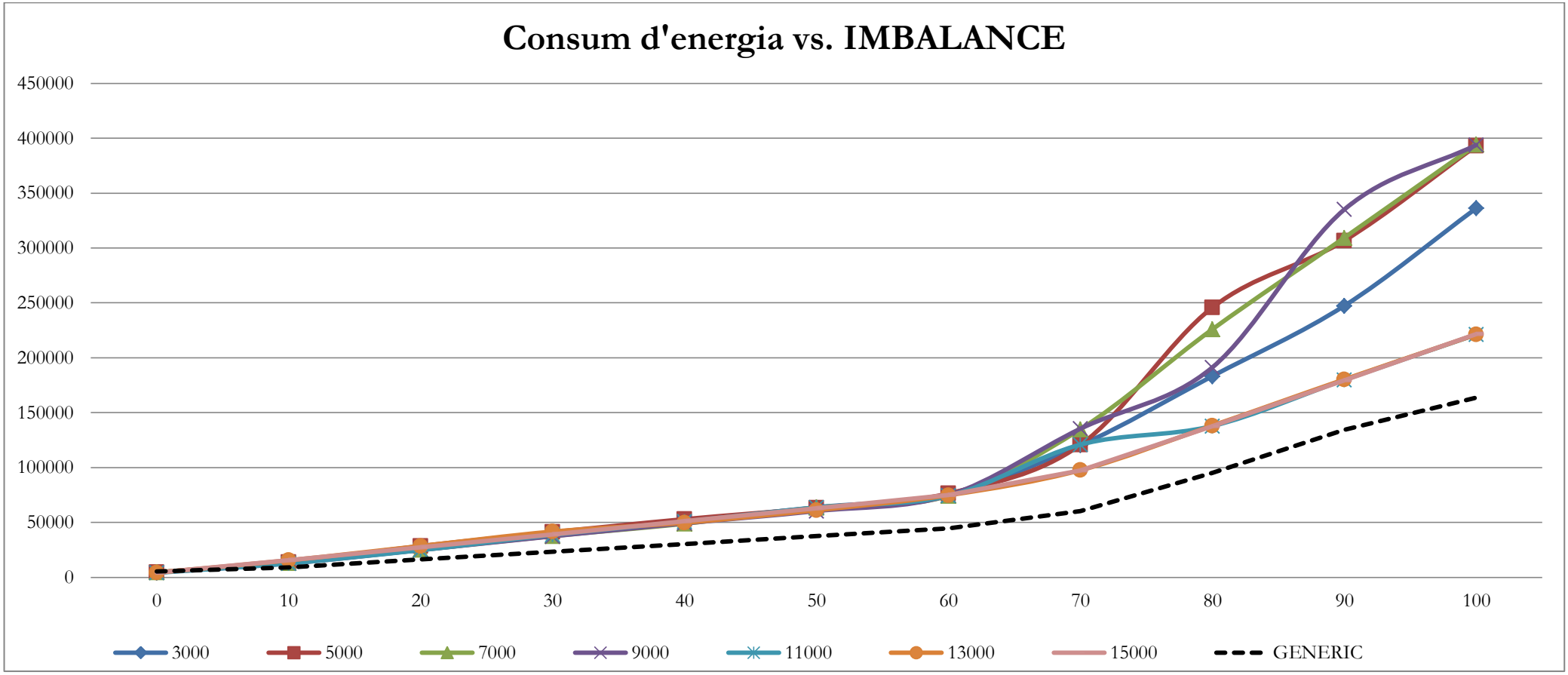


Figura 1: Consum d'energia del programa de multiplicació de matrius utilitzant diferents nivells de desbalanceig.

Les diferents sèries es basen en el temps d'espera aplicat en la barrera asíncrona: 3000 ns en blau fort, 5000 ns en vermell, 7000 ns en verd, 9000 ns en morat, 11000 ns en blau clar, 13000 ns en taronja i 15000 ns en rosa. A més a més, es representa en línia discontinua negra, la sèrie amb els valors del programa síncron de multiplicació de matrius.

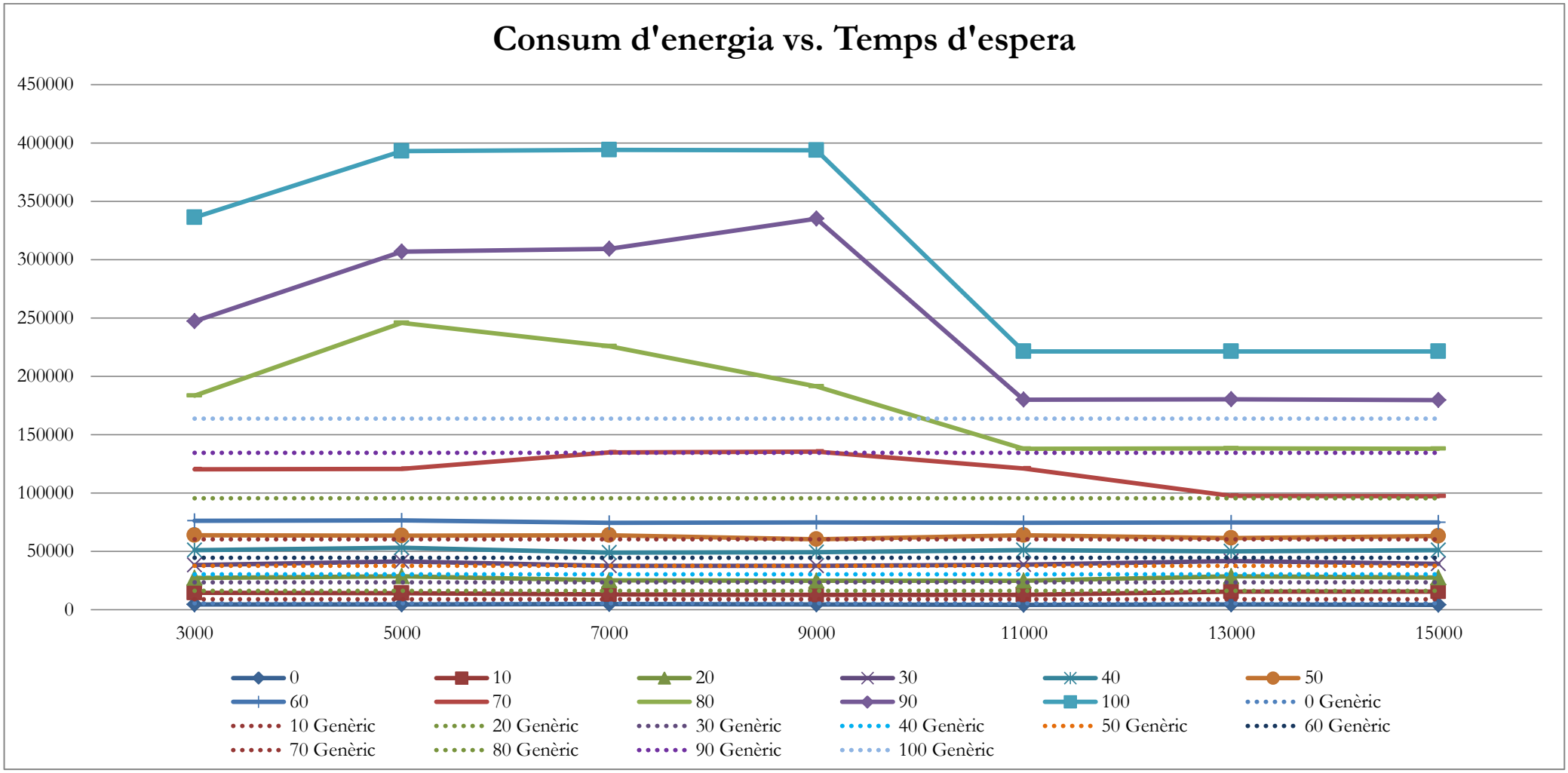


Figura 2: Consum d'energia del programa de multiplicació de matrius utilitzant diferents temps d'espera a la barrera asíncrona.

Les diferents sèries es basen en el desbalanceig aplicat al nombre de càlculs que realitzen els diferents fils d'execució del programa. Les línies continues representes les sèries asíncrones i les línies discontinues les sèries síncrones de resultats. Les sèries síncrones no disposen de marcadors en els punts de valors, ja que es representen com a nivell de referencia per les sèries asíncrones. Els colors són equivalents segons el grau de desbalanceig aplicat. Així tenim el blau amb diamants marcant els punts per un 0%, el color vermell amb quadrats com a marcadors per un 10%, el verd amb triangles marcadors per indicar un 20%, el morat amb ics pel 30%, el blau turquesa amb marcadors ics pel 40%, el taronja pel 50%, el blau amb línies verticals com a marcadors pel 60%, el vermell sense marcadors pel 70%, el verd sense marcadors pel 80%, el morat amb diamants pel 90% i el blau turquesa amb quadrats pel 100%.

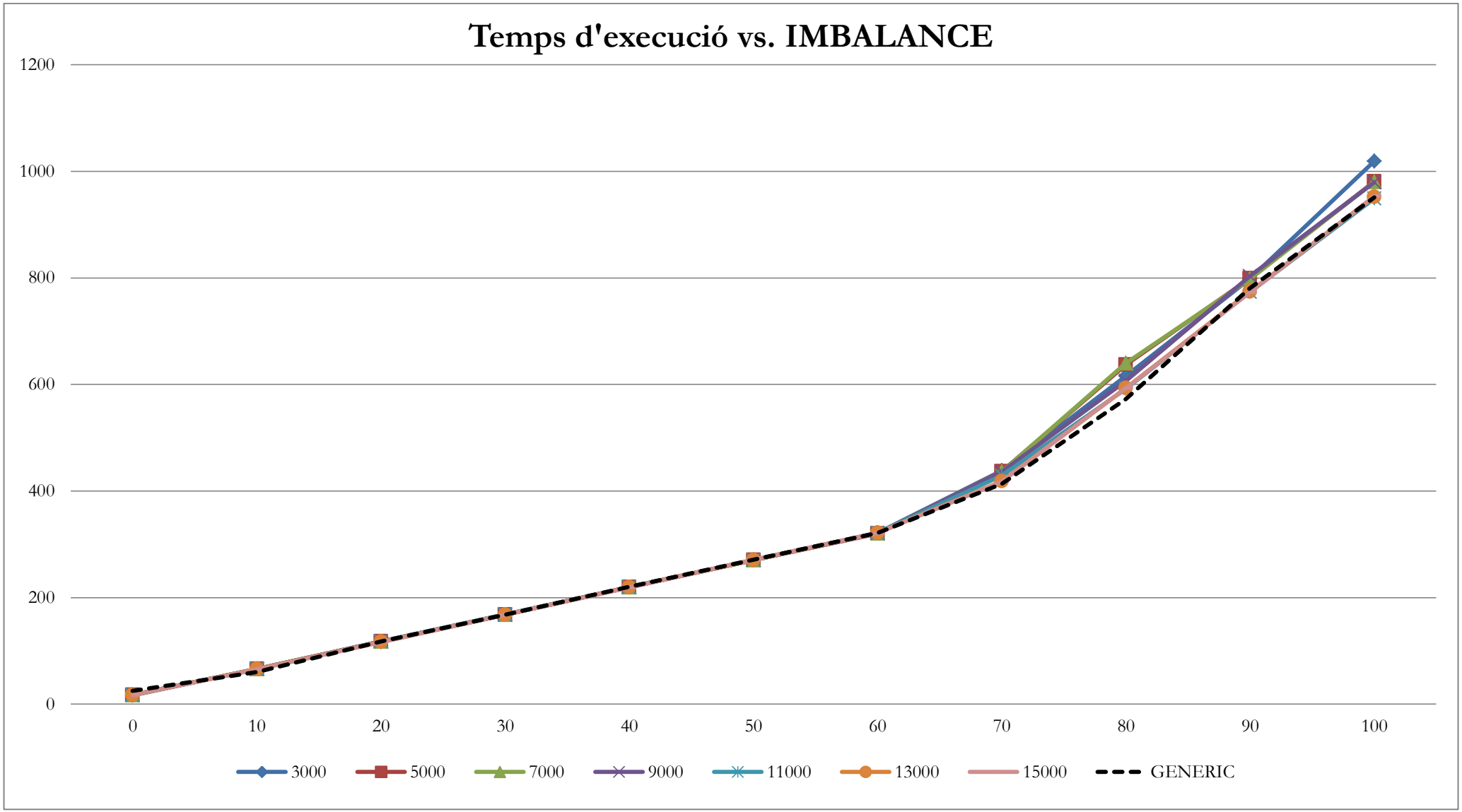


Figura 3: Temps d'execució del programa de multiplicació de matrius utilitzant diferents nivells de desbalanceig.

Les diferents sèries es basen en el temps d'espera aplicat en la barrera asíncrona: 3000 ns en blau fort, 5000 ns en vermell, 7000 ns en verd, 9000 ns en morat, 11000 ns en blau clar, 13000 ns en taronja i 15000 ns en rosa. A més a més, es representa en línia discontinua negra, la sèrie amb els valors del programa síncron de multiplicació de matrius.

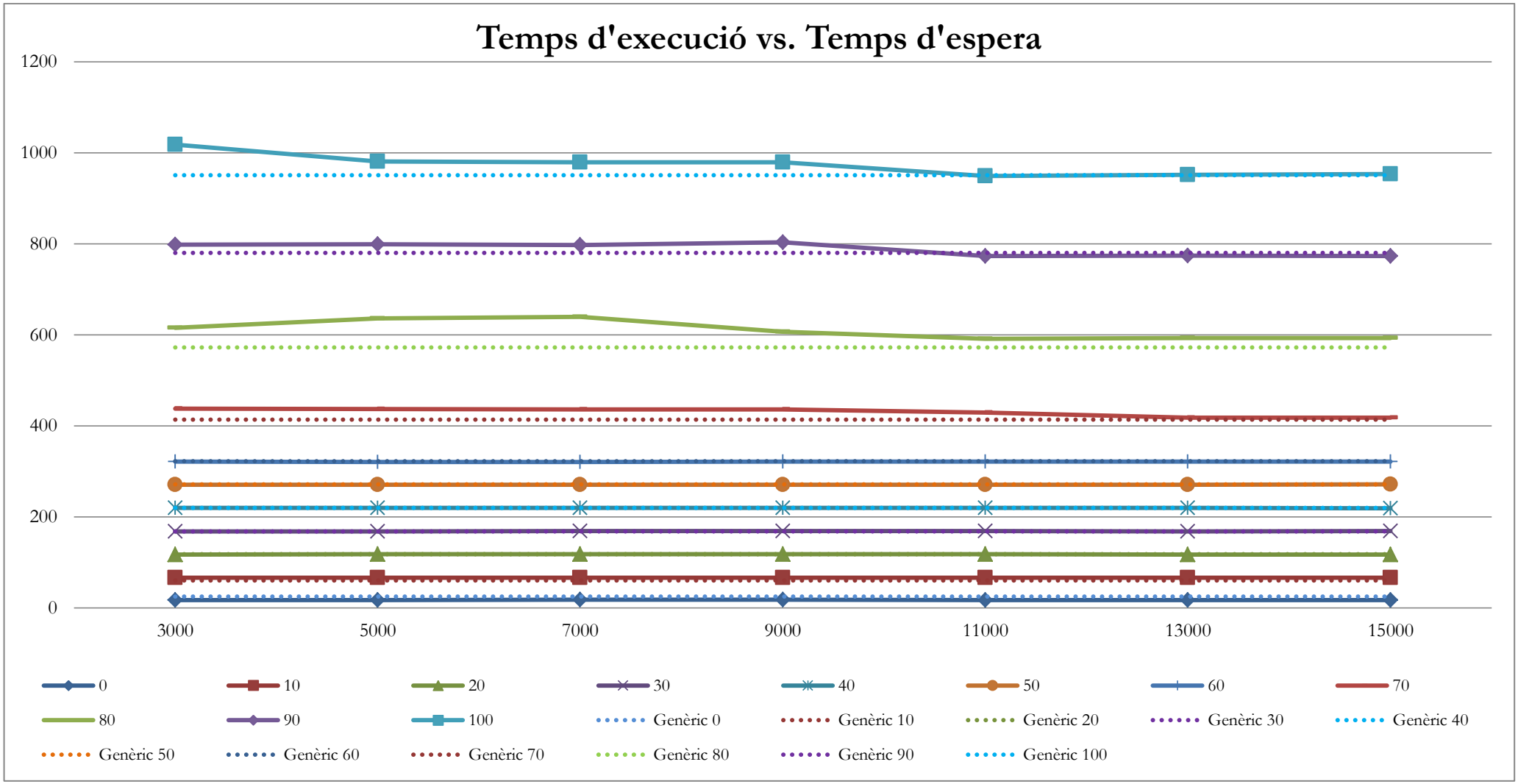


Figura 4: Temps d'execució del programa de multiplicació de matrius utilitzant diferents temps d'espera a la barrera asíncrona.

Les diferents sèries es basen en el desbalanceig aplicat al nombre de càlculs que realitzen els diferents fils d'execució del programa. Les línies continues representes les sèries asíncrones i les línies discontinues les sèries síncrones de resultats. Les sèries síncrones no disposen de marcadors en els punts de valors, ja que es representen com a nivell de referencia per les sèries asíncrones. Els colors són equivalents segons el grau de desbalanceig aplicat. Així tenim el blau marí per un 0%, el color vermell amb quadrats com a marcadors per un 10%, el verd per indicar un 20%, el morat fort pel 30%, el blau turquesa pel 40%, el taronja pel 50%, el blau pel 60%, el vermell amb línia horitzontal com a marcadors pel 70%, el verd sense marcadors pel 80%, el morat clar pel 90% i el blau turquesa pel 100%.

Consum energètic + temps d'execució vs. IMBALANCE

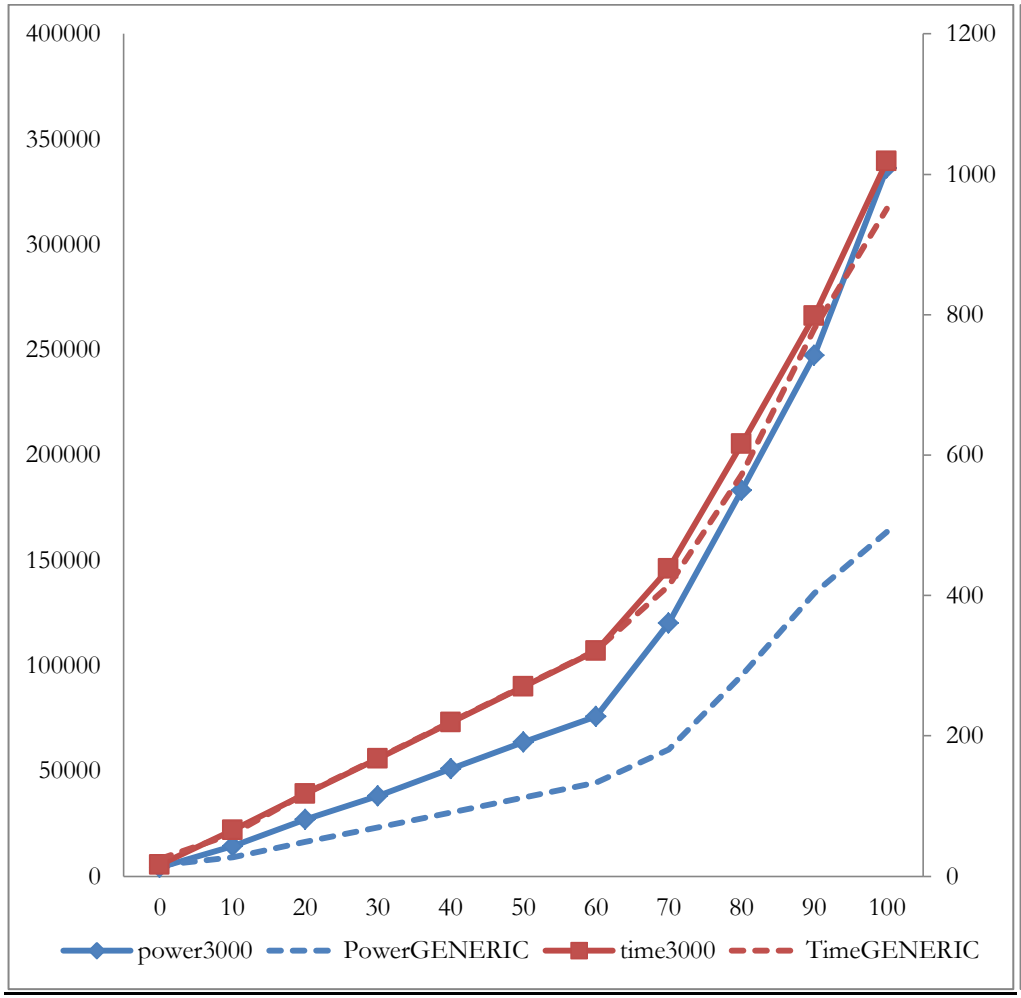


Figura 5: Consum d'energia i temps d'execució del programa de multiplicació de matrius utilitzant diferents nivells de desbalanceig i amb temps d'espera de 3000ns.

Les sèries amb línia continua indiquen els valors dels càlculs asíncron i les línies discontinues els valors corresponen als resultats síncrons. En blau tenim els valors de consum energètic i en vermell els valors de temps d'execució.

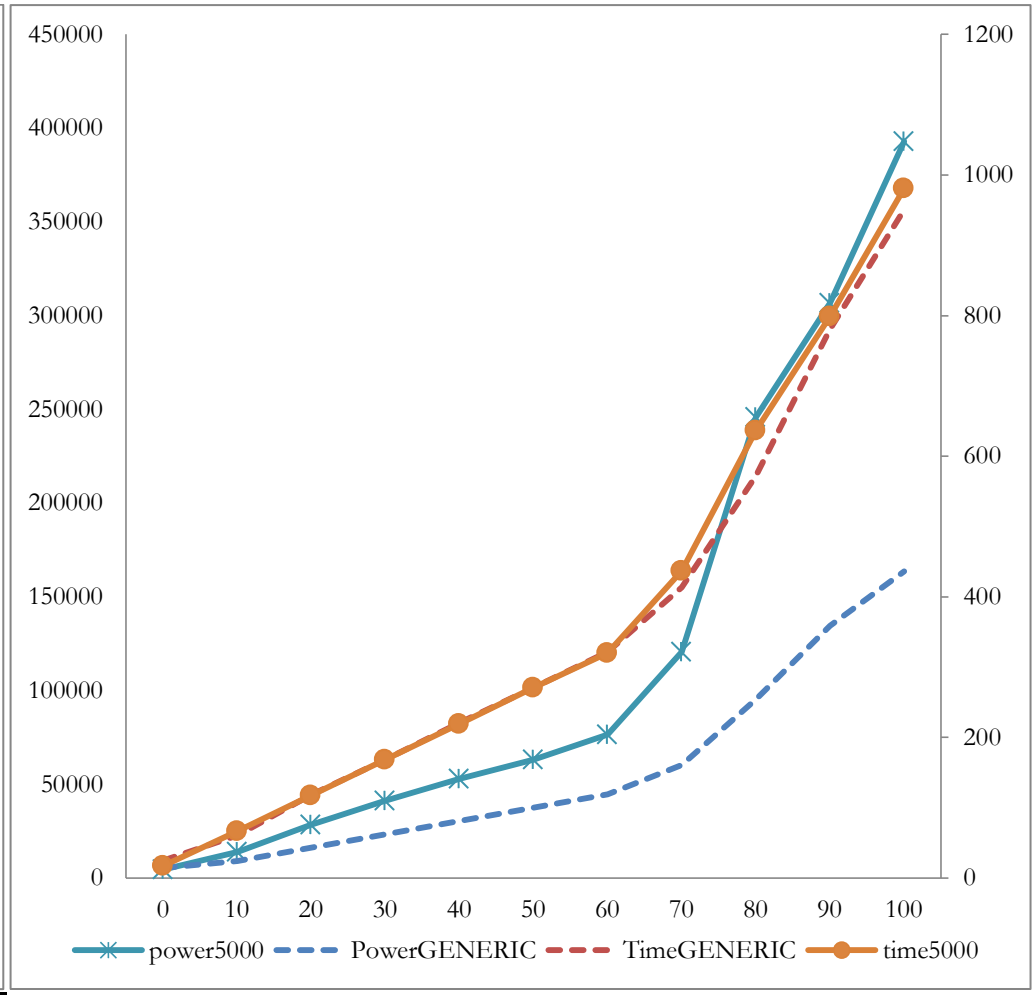


Figura 6: Consum d'energia i temps d'execució del programa de multiplicació de matrius utilitzant diferents nivells de desbalanceig i amb temps d'espera de 5000ns.

Les sèries mostrades en línia continua corresponen als valors asíncrons: en blau els consums energètics i el taronja els temps d'execució. Les sèries amb línia discontinua corresponen als valors síncrons: en blau el consum d'energia i en vermell el temps d'execució

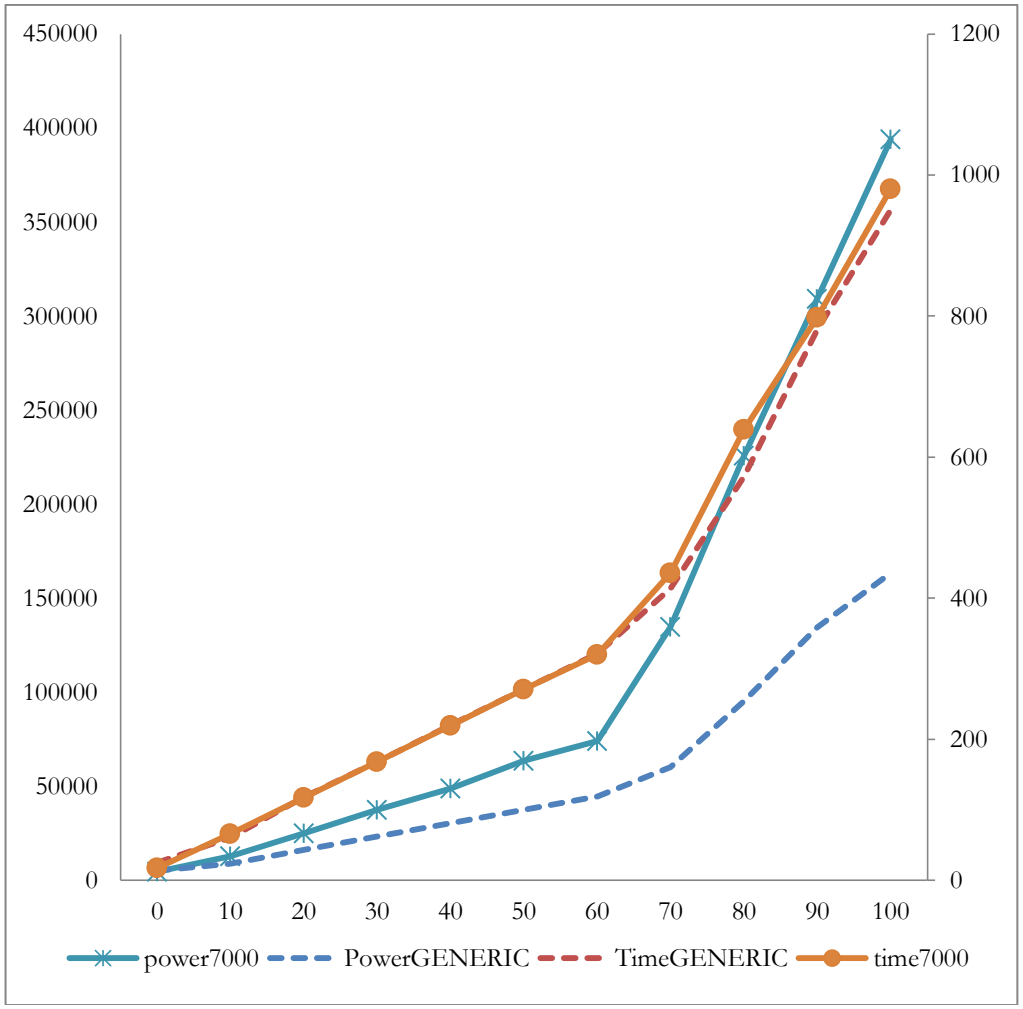


Figura 7: Consum d'energia i temps d'execució del programa de multiplicació de matrius utilitzant diferents nivells de desbalanceig i amb temps d'espera de 7000ns.

Les sèries mostrades en línia continua corresponen als valors asíncrons: en blau els consums energètics i el taronja els temps d'execució. Les sèries amb la línia discontinua corresponen als valors síncrons: en blau el consum d'energia i en vermell el temps d'execució.

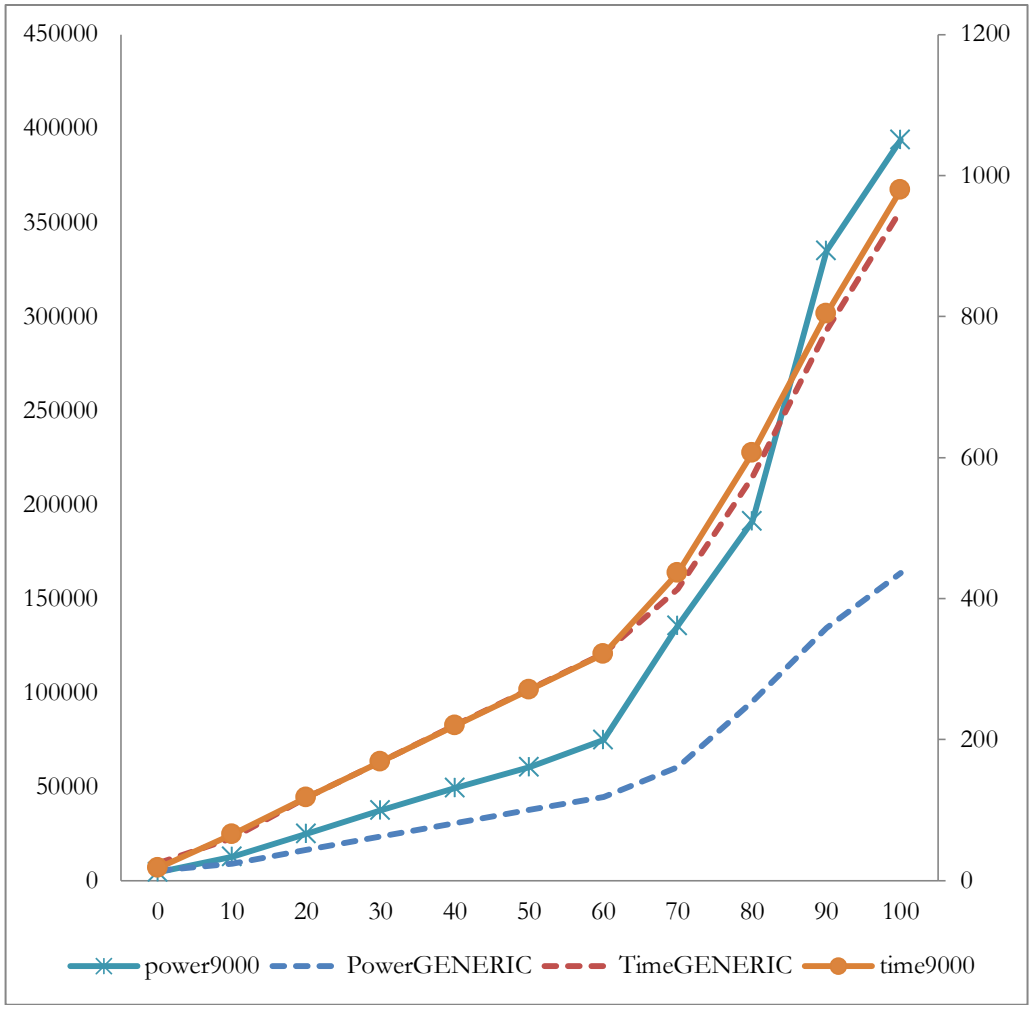


Figura 8: Consum d'energia i temps d'execució del programa de multiplicació de matrius utilitzant diferents nivells de desbalanceig i amb temps d'espera de 9000ns

Les sèries mostrades en línia continua corresponen als valors asíncrons: en blau els consums energètics i el taronja els temps d'execució. Les sèries amb la línia discontinua corresponen als valors síncrons: en blau el consum d'energia i en vermell el temps d'execució.

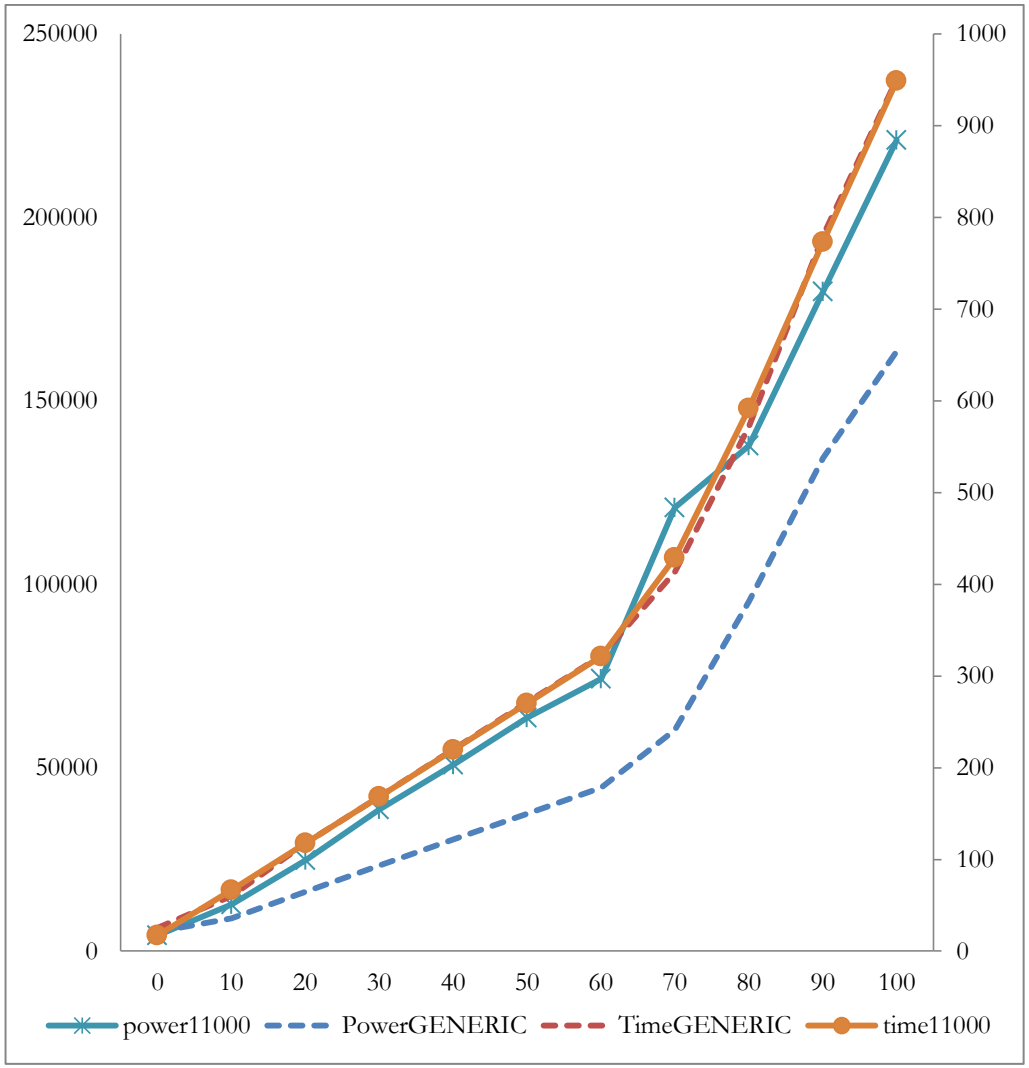


Figura 9: Consum d'energia i temps d'execució del programa de multiplicació de matrius utilitzant diferents nivells de desbalanceig i amb temps d'espera d'11000ns.

Les sèries mostrades en línia continua corresponen als valors asíncrons: en blau els consums energètics i el taronja els temps d'execució. Les sèries amb la línia discontinua corresponen als valors síncrons: en blau el consum d'energia i en vermell el temps d'execució.

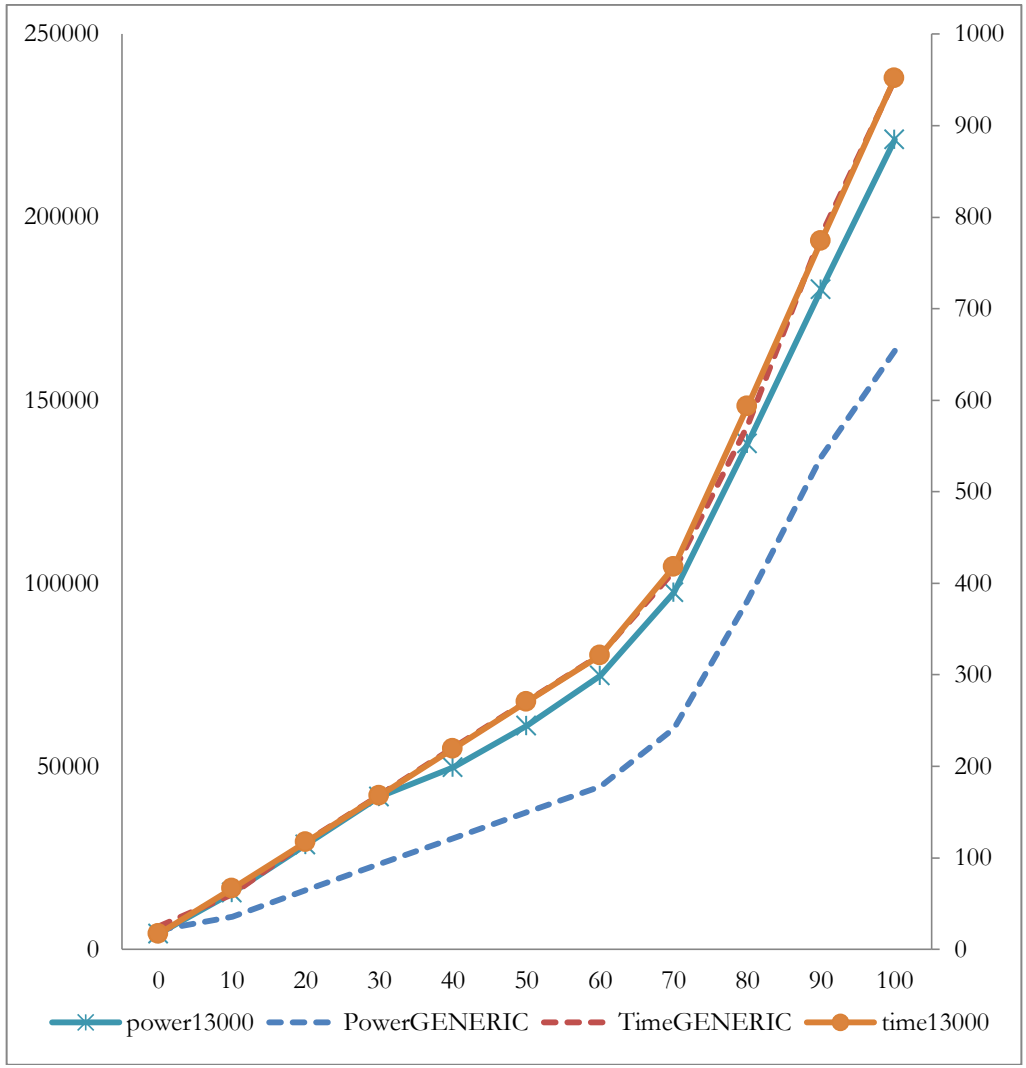


Figura 10: Consum d'energia i temps d'execució del programa de multiplicació de matrius utilitzant diferents nivells de desbalanceig i amb temps d'espera de 13000ns.

Les sèries mostrades en línia continua corresponen als valors asíncrons: en blau els consums energètics i el taronja els temps d'execució. Les sèries amb la línia discontinua corresponen als valors síncrons: en blau el consum d'energia i en vermell el temps d'execució.

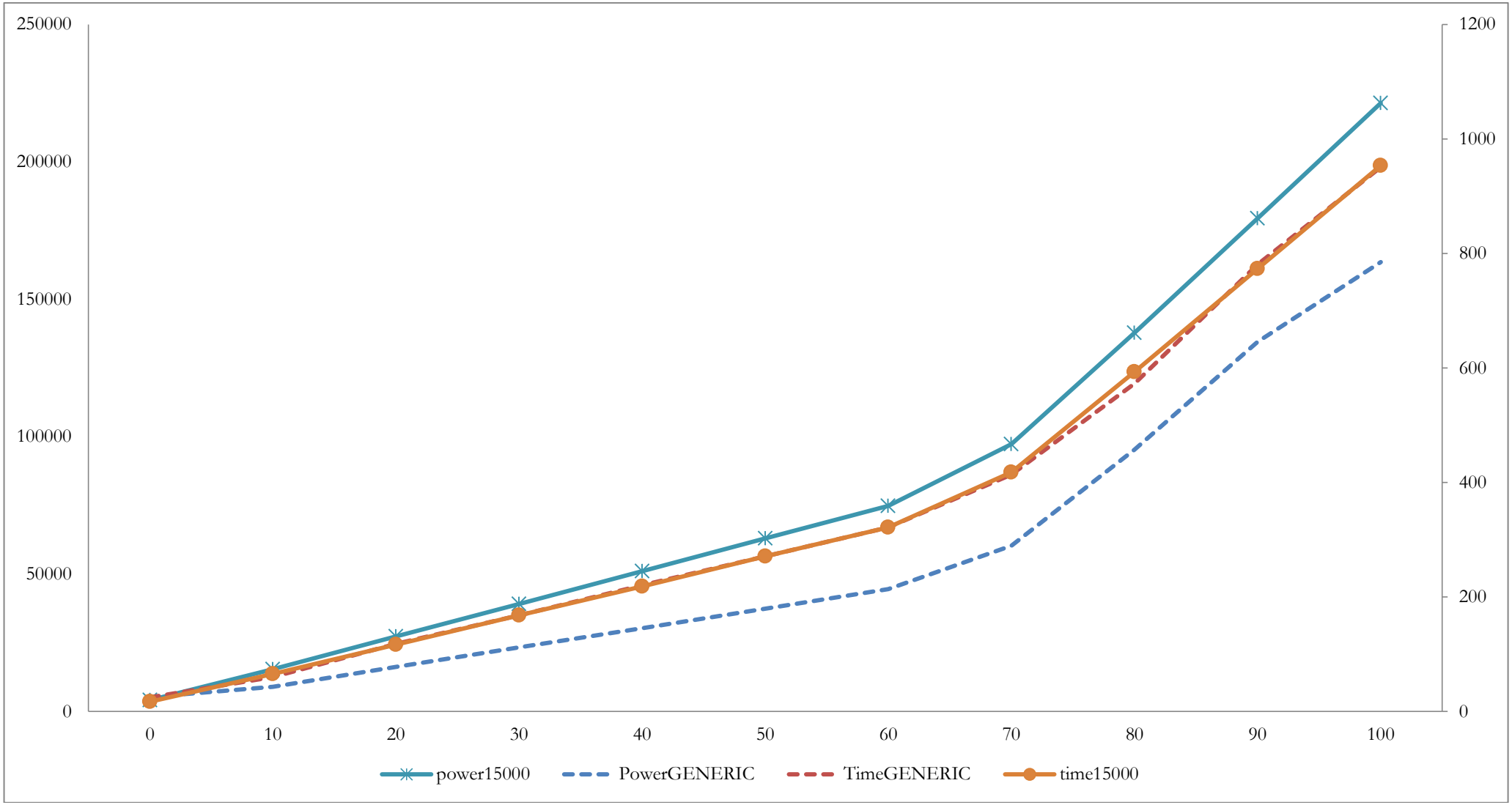


Figura 11: Consum d'energia i temps d'execució del programa de multiplicació de matrius utilitzant diferents nivells de desbalanceig i amb temps d'espera de 15000ns.

Les sèries mostrades en línia continua corresponen als valors asíncrons: en blau els consums energètics i el taronja els temps d'execució. Les sèries amb la línia discontinua corresponen als valors síncrons: en blau el consum d'energia i en vermell el temps d'execució.

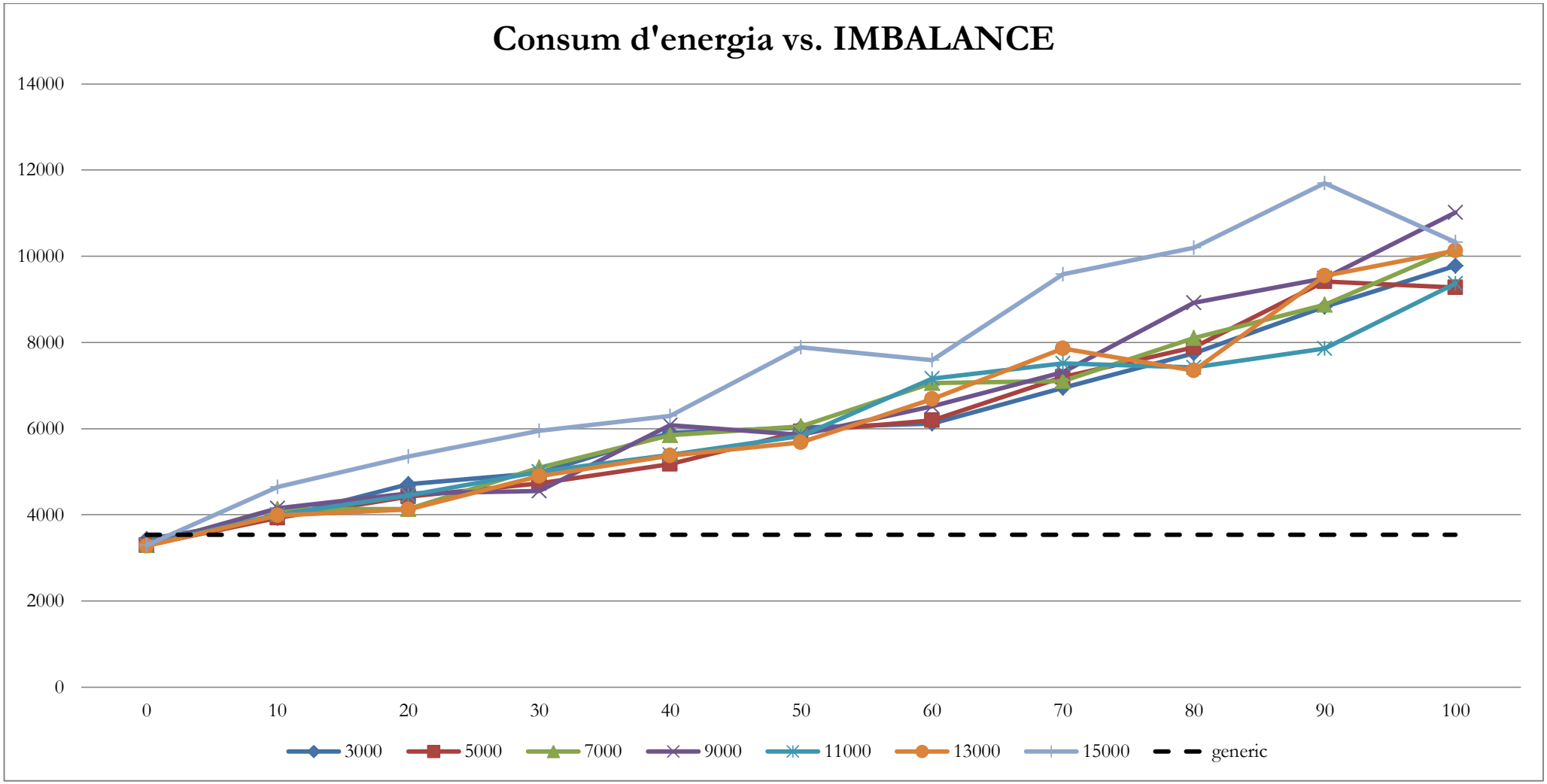


Figura 12: Consum d'energia del programa de càlcul del número π utilitzant diferents nivells de desbalanceig.

Les diferents sèries es basen en el temps d'espera aplicat en la barrera asíncrona: 3000ns en blau fort, 5000ns en vermell, 7000ns en verd, 9000ns en morat, 11000ns en blau clar, 13000ns en taronja i 15000ns en morat clar. A més a més, es representa en línia discontinua negra, la sèrie amb els valors del programa síncron de càlcul del número π.

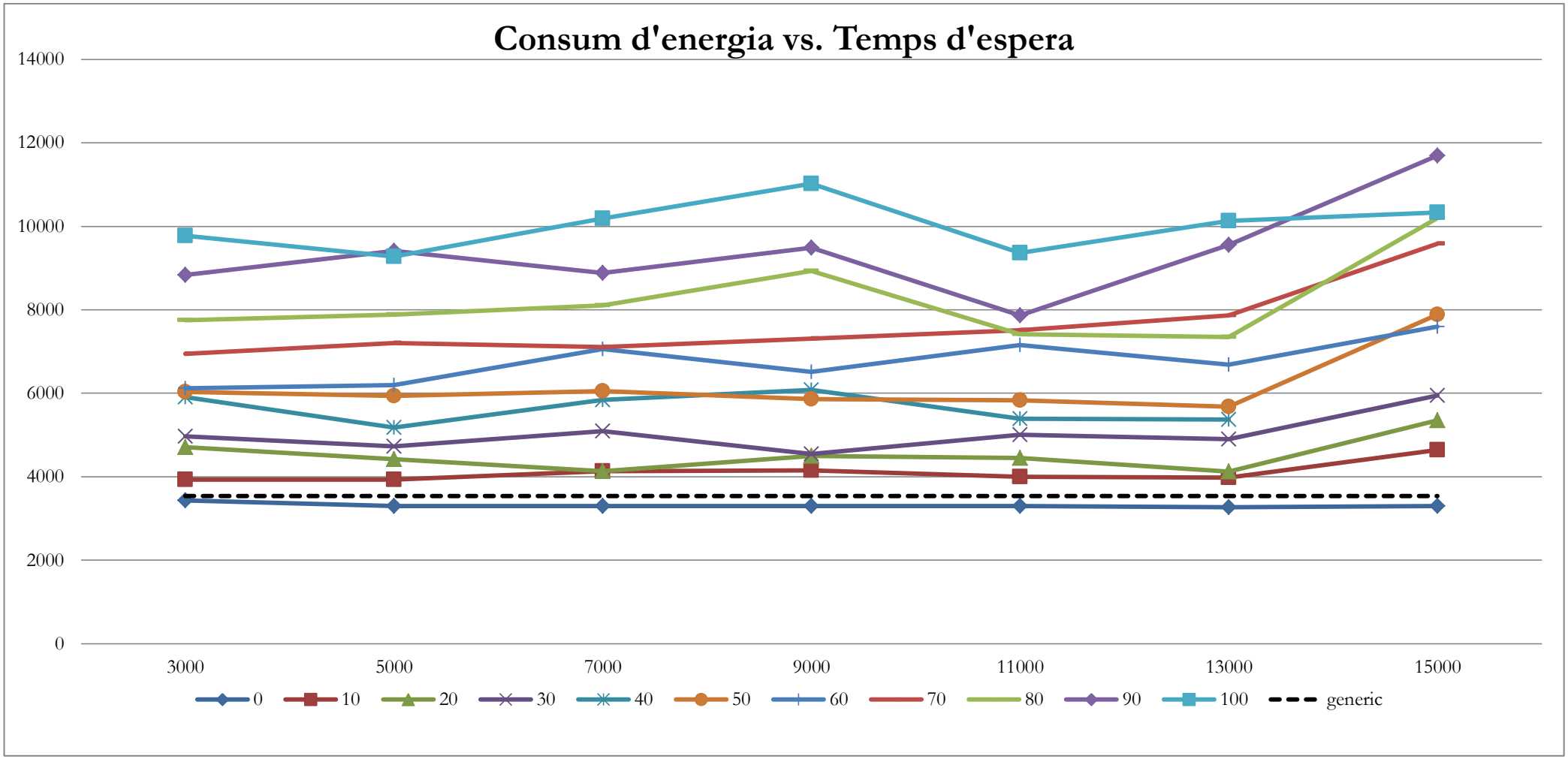


Figura 13: Consum d'energia del programa de càlcul del número π utilitzant diferents temps d'espera a la barrera asíncrona.

Les diferents sèries es basen en el desbalanceig aplicat al nombre de càlculs que realitzen els diferents fils d'execució del programa. La línia discontinua negra la sèrie síncrona de resultats i les línies continues representes les sèries asíncrones. Així tenim el blau amb diamants marcant els punts per un 0%, el color vermell amb quadrats com a marcadors per un 10%, el verd amb triangles marcadors per indicar un 20%, el morat amb ics pel 30%, el blau turquesa amb marcadors ics pel 40%, el taronja pel 50%, el blau amb línies verticals com a marcadors pel 60%, el vermell sense marcadors pel 70%, el verd sense marcadors pel 80%, el morat amb diamants pel 90% i el blau turquesa amb quadrats pel 100%.

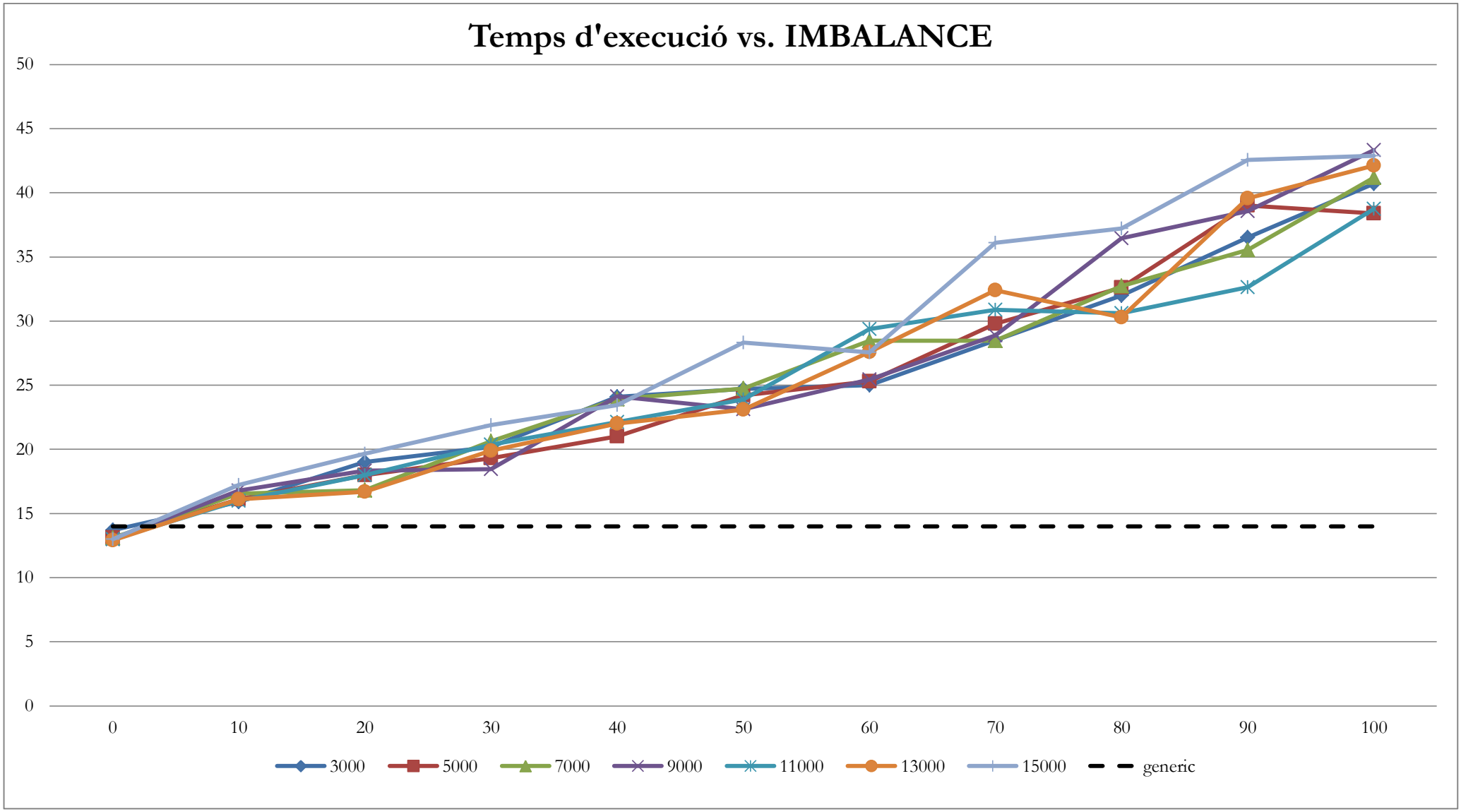


Figura 14: Temps d'execució del programa de càlcul del número π utilitzant diferents nivells de desbalanceig.

Les diferents sèries es basen en el temps d'espera aplicat en la barrera asíncrona: 3000ns en blau fort, 5000ns en vermell, 7000ns en verd, 9000ns en morat, 11000ns en blau clar, 13000ns en taronja i 15000ns en morat clar. A més a més, es representa en línia discontinua negra, la sèrie amb els valors del programa síncron de càlcul del número π.

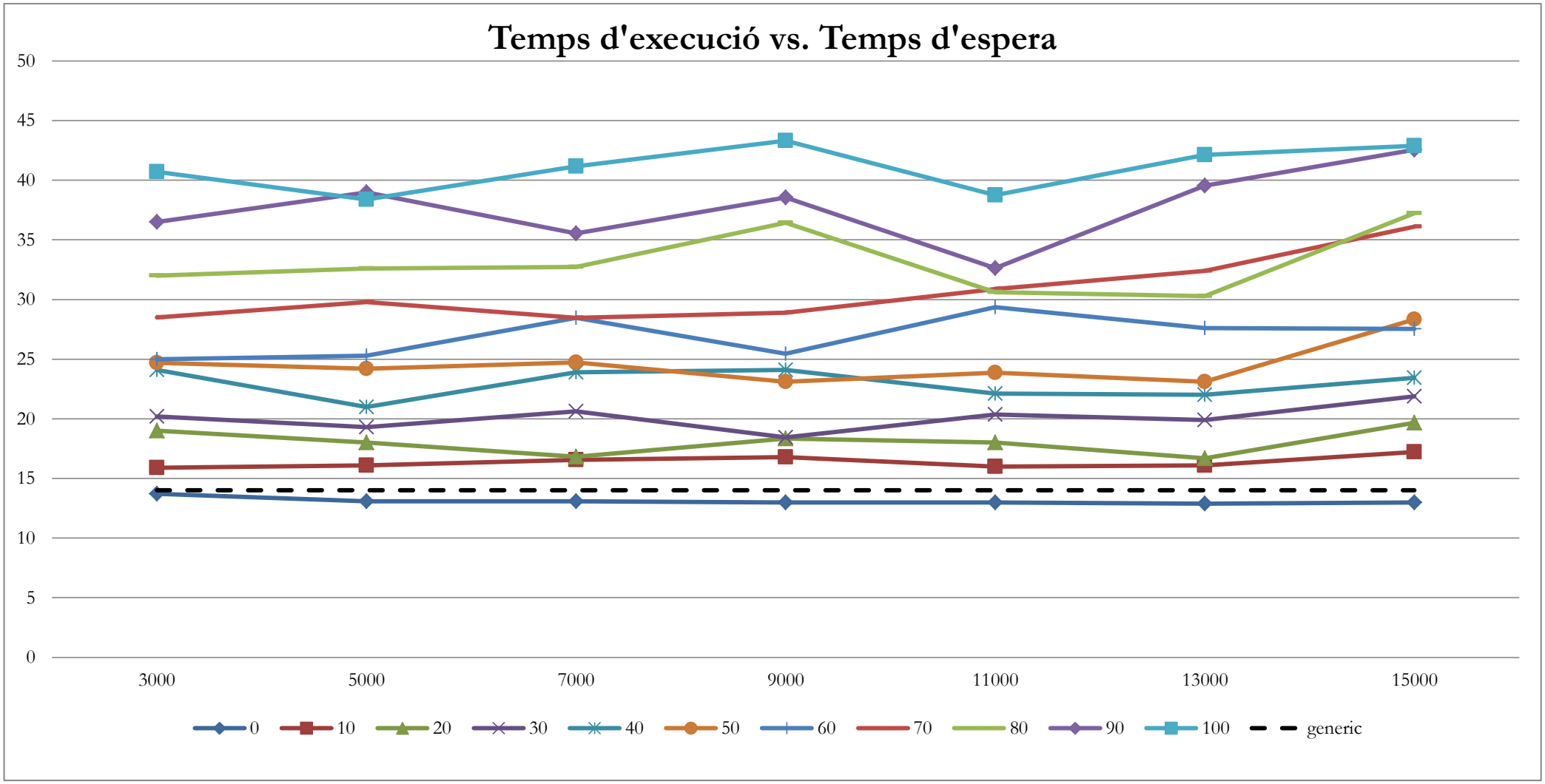


Figura 15: Temps d'execució del programa de càlcul del número π utilitzant diferents temps d'espera a la barrera asíncrona.

Les diferents sèries es basen en el desbalanceig aplicat al nombre de càlculs que realitzen els diferents fils d'execució del programa. Les línies continues representes les sèries asíncrones s. Els colors són equivalents segons el grau de desbalanceig aplicat. Així tenim el blau marí per un 0%, el color vermell amb quadrats com a marcadors per un 10%, el verd amb triangles per indicar un 20%, el morat fort pel 30%, el blau turquesa pel 40%, el taronja pel 50%, el blau pel 60%, el vermell sense marcadors pel 70%, el verd sense marcadors pel 80%, el morat amb diamants pel 90% i el blau turquesa amb marcadors quadrats pel 100%. Com a valor de referencia tenim marcada amb línia discontinua la sèrie de resultats síncrons.

Consum energètic + temps d'execució vs. IMBALANCE

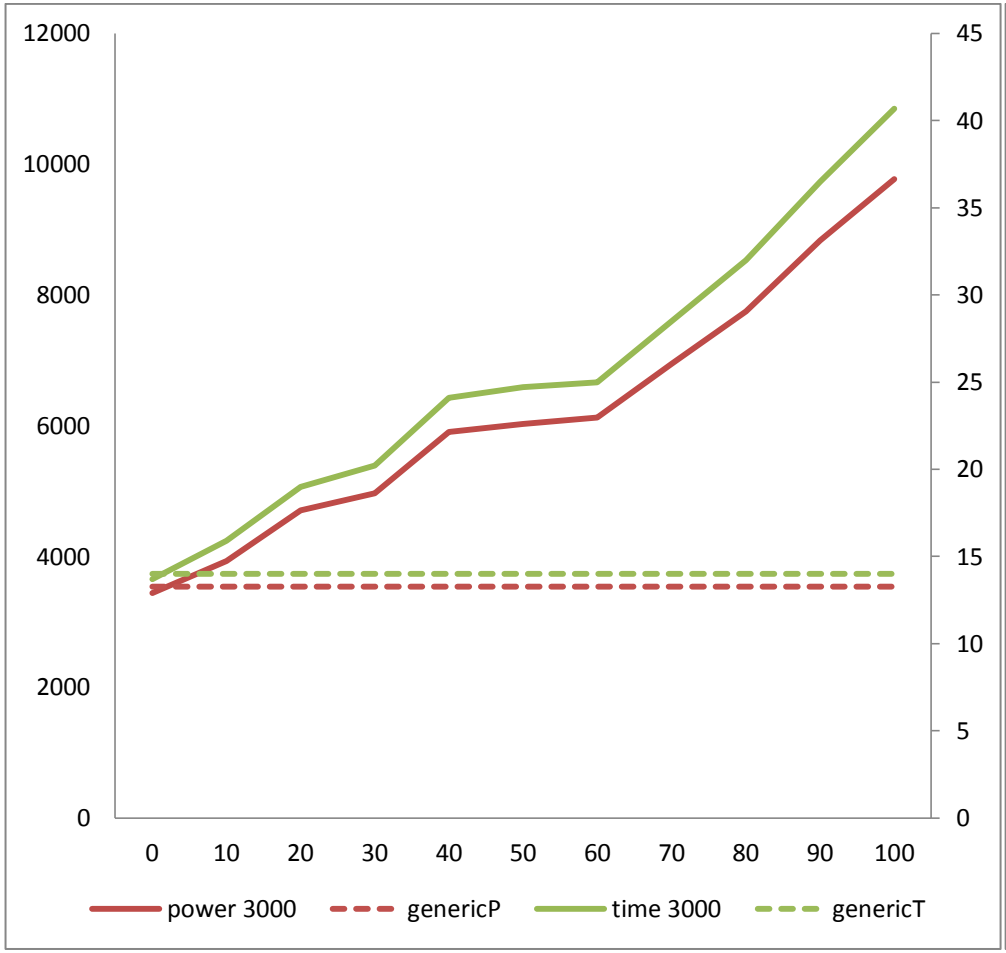


Figura 16: Consum d'energia i temps d'execució del programa de càlcul del número π utilitzant diferents nivells de desbalanceig i amb temps d'espera de 3000ns.

Les sèries amb línia continua indiquen els valors dels càlculs asíncron i les línies discontinues els valors corresponen als resultats síncrons. En vermell tenim els valors de consum energètic i en verd els valors de temps d'execució.

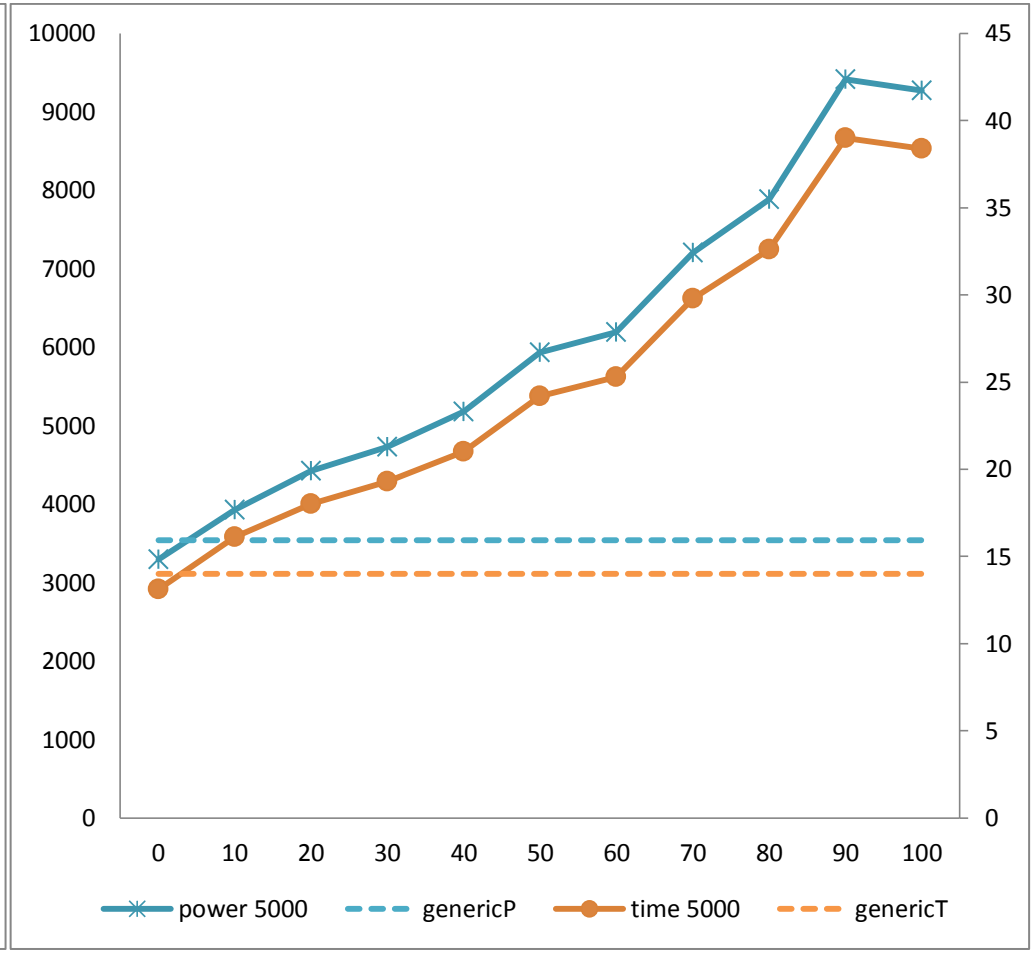


Figura 17: Consum d'energia i temps d'execució del programa de càlcul del número π utilitzant diferents nivells de desbalanceig i amb temps d'espera de 5000ns.

Les sèries amb línia continua indiquen els valors dels càlculs asíncron i les línies discontinues els valors corresponen als resultats síncrons. En blau tenim els valors de consum energètic i en taronja els valors de temps d'execució.

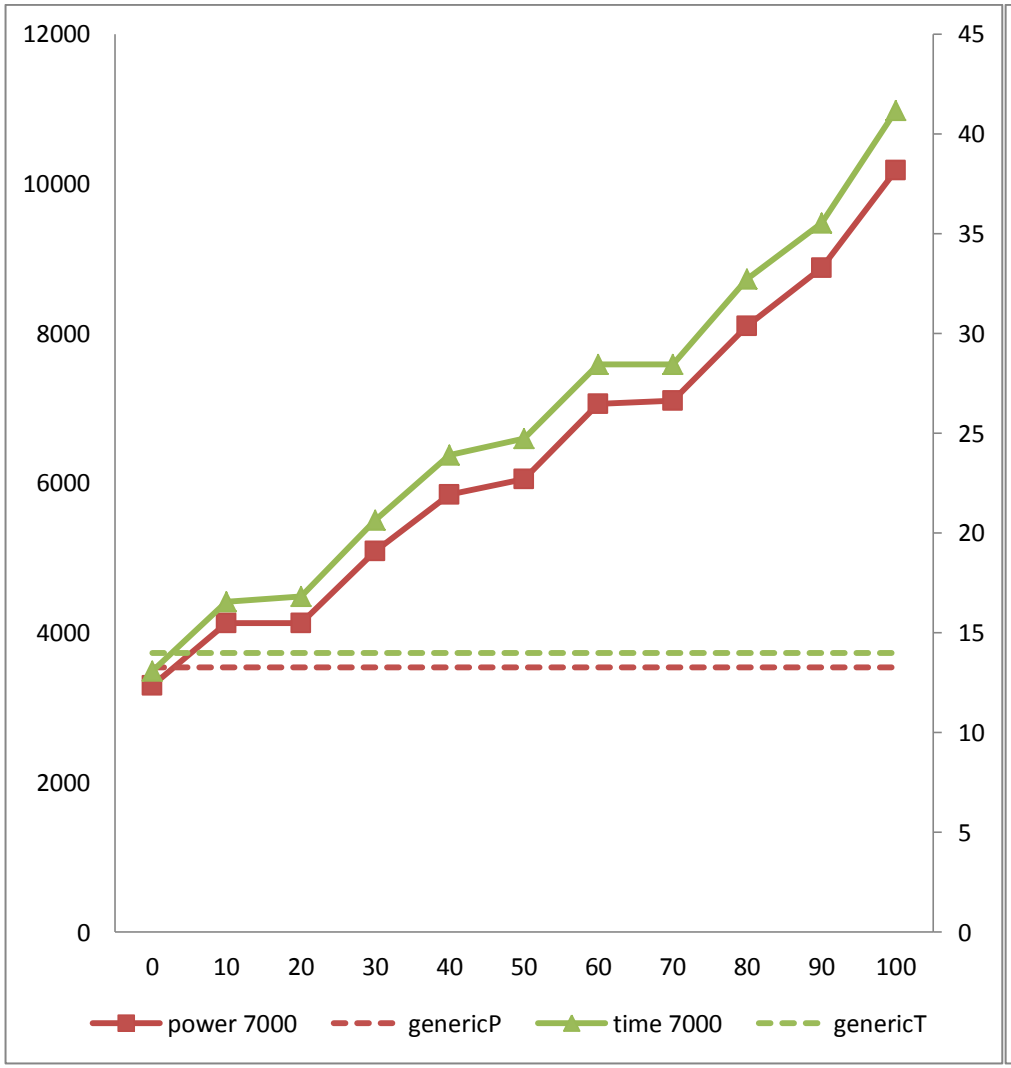


Figura 18: Consum d'energia i temps d'execució del programa de càlcul del número π utilitzant diferents nivells de desbalanceig i amb temps d'espera de 7000ns.

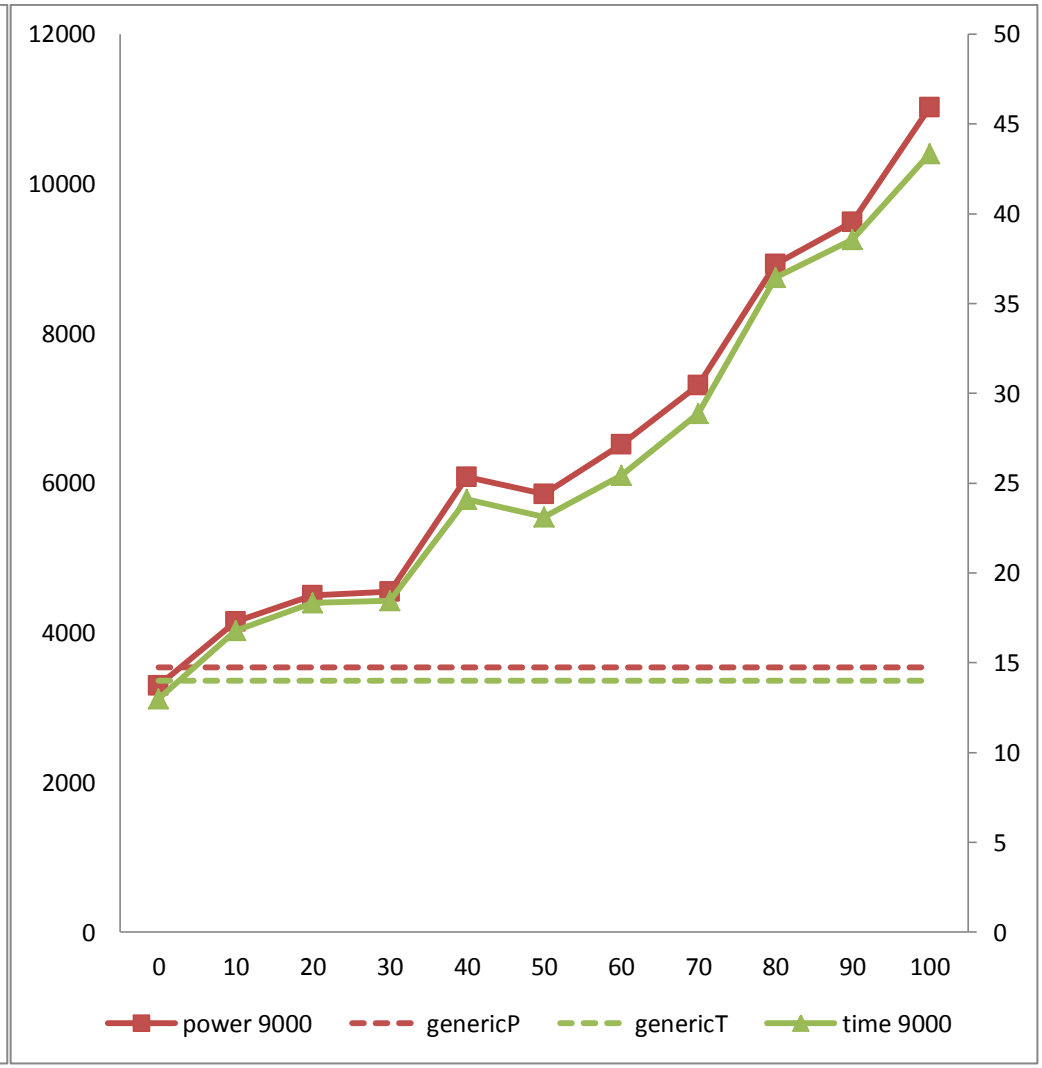


Figura 19: Consum d'energia i temps d'execució del programa de càlcul del número π utilitzant diferents nivells de desbalanceig i amb temps d'espera de 9000ns.

Les sèries amb línia continua indiquen els valors dels càlculs asíncron i les línies discontinues els valors corresponen als resultats síncrons. En vermell tenim els valors de consum energètic i en verd els valors de temps d'execució.

Les sèries amb línia continua indiquen els valors dels càlculs asíncron i les línies discontinues els valors corresponen als resultats síncrons. En vermell tenim els valors de consum energètic i en verd els valors de temps d'execució.

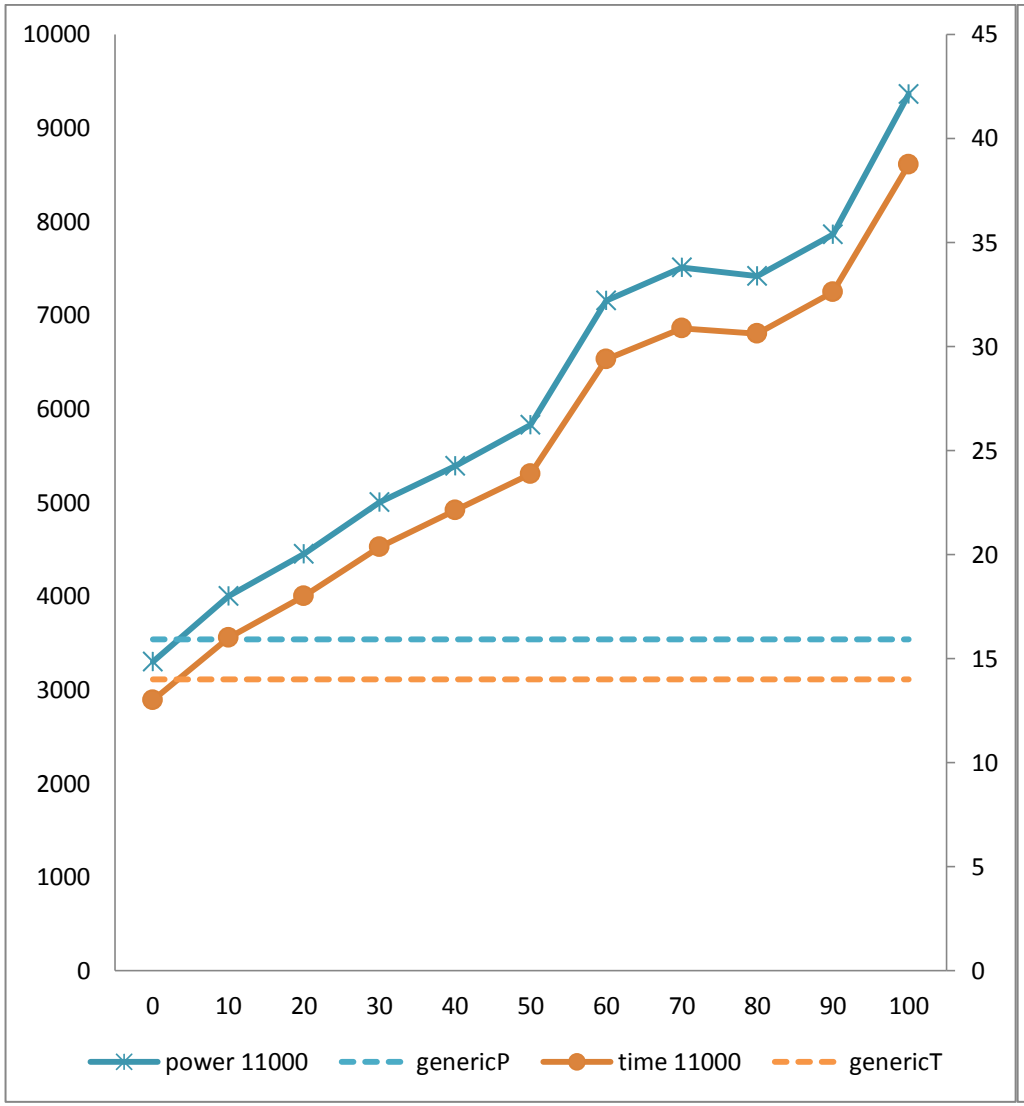


Figura 20: Consum d'energia i temps d'execució del programa de càlcul del número π utilitzant diferents nivells de desbalanceig i amb temps d'espera d'11000ns.

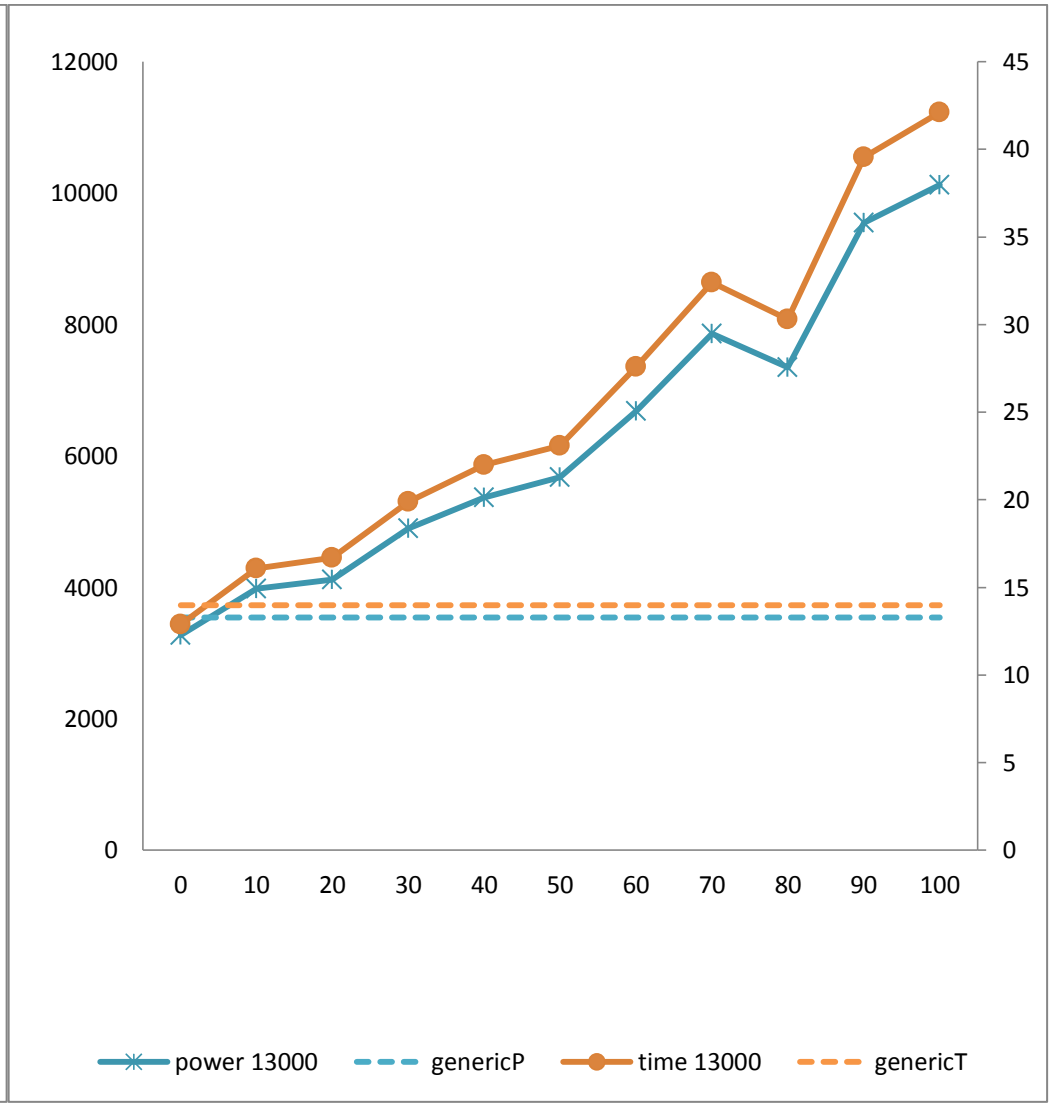


Figura 21: Consum d'energia i temps d'execució del programa de càlcul del número π utilitzant diferents nivells de desbalanceig i amb temps d'espera de 13000ns.

Les sèries amb línia continua indiquen els valors dels càlculs asíncron i les línies discontinues els valors corresponen als resultats síncrons. En blau tenim els valors de consum energètic i en taronja els valors de temps d'execució.

Les sèries amb línia continua indiquen els valors dels càlculs asíncron i les línies discontinues els valors corresponen als resultats síncrons. En blau tenim els valors de consum energètic i en taronja els valors de temps d'execució.

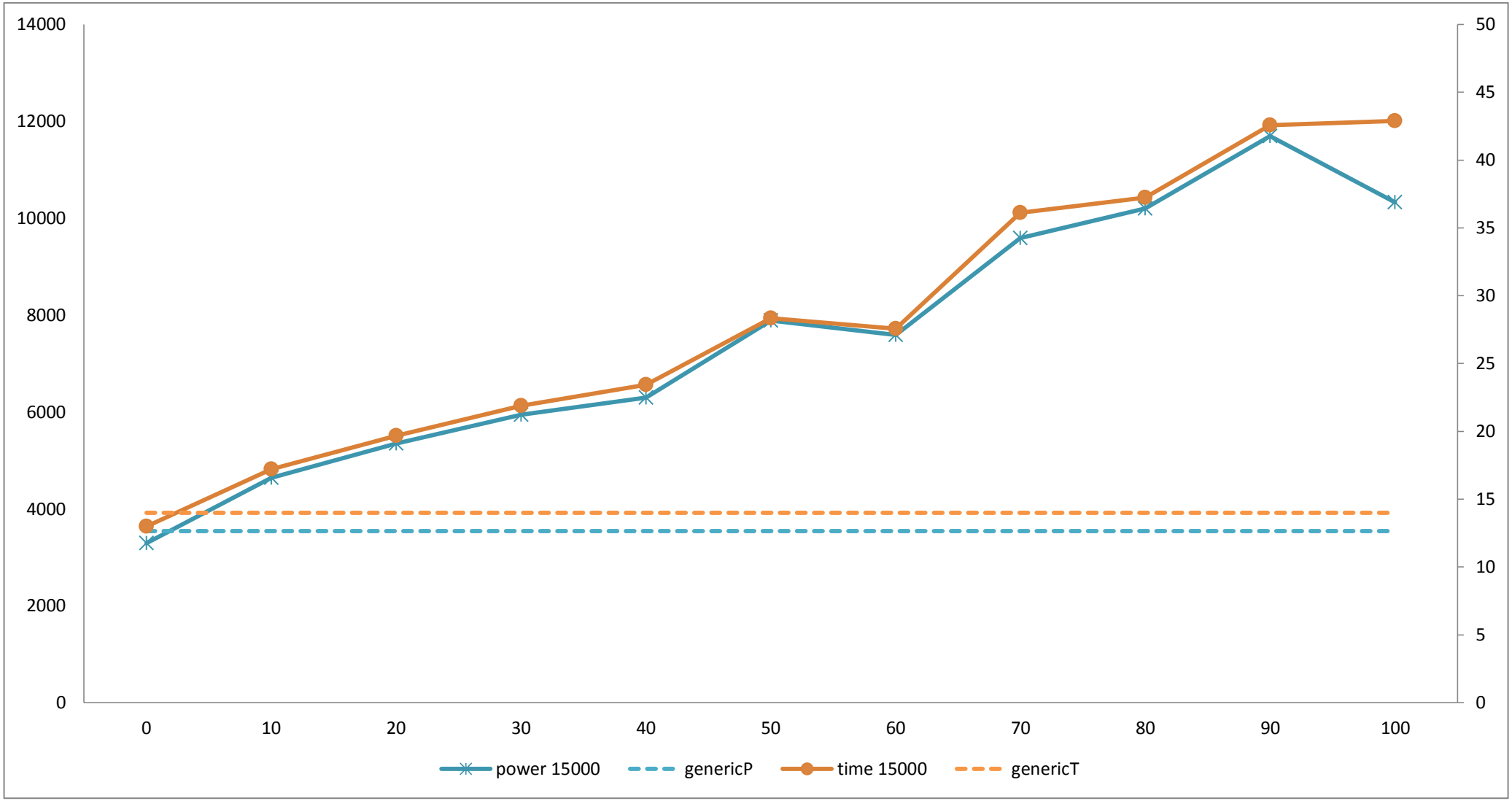


Figura 22: Consum d'energia i temps d'execució del programa de càlcul del número π utilitzant diferents nivells de desbalanceig i amb temps d'espera de 15000ns.

Les sèries amb línia continua indiquen els valors dels càlculs asíncron i les línies discontinues els valors corresponen als resultats síncrons. En blau tenim els valors de consum energètic i en taronja els valors de temps d'execució

11 Codi programes

11.1 Mutiplicació de matrius

11.1.1 Síncron

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

//#define NRA 20                 // number of rows in matrix A
//#define NCA 20                 // number of columns in matrix A
//#define NCB 20                 // number of columns in matrix B

//#define SIZE 100
//#define ITERS 10
//#define IMBALANCE 10 // 0 is not imbalance
#define IMBALANCE_BASE 50

void matmul(int my_size){

  int   i, j, k;
  double **a, **b, **c;
  double *a_block, *b_block, *c_block;
  double **res;
  double *res_block;
  int NRA, NCA, NCB;

  NRA = my_size; NCA = my_size; NCB = my_size;

  a = (double **) malloc(NRA*sizeof(double *)); /* matrix a to be multiplied */
  b = (double **) malloc(NCA*sizeof(double *)); /* matrix b to be multiplied */
  c = (double **) malloc(NRA*sizeof(double *)); /* result matrix c */

  a_block = (double *) malloc(NRA*NCA*sizeof(double)); /* Storage for matrices */
  b_block = (double *) malloc(NCA*NCB*sizeof(double));
  c_block = (double *) malloc(NRA*NCB*sizeof(double));

  /* Result matrix for the sequential algorithm */
  res = (double **) malloc(NRA*sizeof(double *));
  res_block = (double *) malloc(NRA*NCB*sizeof(double));

  for (i=0; i<NRA; i++)   /* Initialize pointers to a */
    a[i] = a_block+i*NRA;

  for (i=0; i<NCA; i++)   /* Initialize pointers to b */
    b[i] = b_block+i*NCA;

  for (i=0; i<NRA; i++)   /* Initialize pointers to c */
    c[i] = c_block+i*NRA;

  for (i=0; i<NRA; i++)   /* Initialize pointers to res */
    res[i] = res_block+i*NRA;

    for (i=0; i<NRA; i++) /* last matrix has been initialized */
      for (j=0; j<NCA; j++)
        a[i][j]= (double) (i+j);
    for (i=0; i<NCA; i++)
      for (j=0; j<NCB; j++)
        b[i][j]= (double) (i*j);
    for (i=0; i<NRA; i++)
      for (j=0; j<NCB; j++)
        c[i][j]= 0.0;

    for (i=0; i<NRA; i++) {
      for(j=0; j<NCB; j++) {
        for (k=0; k<NCA; k++) {
```

```
          c[i][j] += a[i][k] * b[k][j];
        }
      }
    }
}

int main (int argc, char *argv[]) {

  int MyProc, tag=1, i, j;
  MPI_Status *status;

  int SIZE, IMBALANCE, ITERS;

  if (argc == 4)
  {
   SIZE = strtol(argv[1],NULL,0);
   IMBALANCE = strtol(argv[2],NULL,0);
   ITERS = strtol(argv[3],NULL,0);
  }
  else
  {
  SIZE = 100;
  IMBALANCE = 10;
  ITERS = 10;
  }

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &MyProc);


  for(i=0; i<ITERS; i++){
        for(j=0; j<IMBALANCE_BASE+IMBALANCE*MyProc; j++){
                matmul(SIZE);
        }
        MPI_Barrier(MPI_COMM_WORLD);
        if(MyProc == 0){
                printf("Iteration %d done!\n", i);
        }
  }

  MPI_Finalize();

  if(MyProc == 0){
        printf("Done.\n");
  }

  exit(0);
}
```

11.1.2 Asíncron

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define CLOCK CLOCK_MONOTONIC

//#define NRA 20                // number of rows in matrix A
//#define NCA 20                // number of columns in matrix A
//#define NCB 20                // number of columns in matrix B

//#define SIZE 100
//#define ITERS 10
//#define IMBALANCE 10 // 0 is not imbalance
#define IMBALANCE_BASE 50

void matmul(int my_size){

  int   i, j, k;
  double **a, **b, **c;
  double *a_block, *b_block, *c_block;
```

```
  double **res;
  double *res_block;
  int NRA, NCA, NCB;

  NRA = my_size; NCA = my_size; NCB = my_size;

  a = (double **) malloc(NRA*sizeof(double *)); /* matrix a to be multiplied */
  b = (double **) malloc(NCA*sizeof(double *)); /* matrix b to be multiplied */
  c = (double **) malloc(NRA*sizeof(double *)); /* result matrix c */

  a_block = (double *) malloc(NRA*NCA*sizeof(double)); /* Storage for matrices */
  b_block = (double *) malloc(NCA*NCB*sizeof(double));
  c_block = (double *) malloc(NRA*NCB*sizeof(double));

  /* Result matrix for the sequential algorithm */
  res = (double **) malloc(NRA*sizeof(double *));
  res_block = (double *) malloc(NRA*NCB*sizeof(double));

  for (i=0; i<NRA; i++)   /* Initialize pointers to a */
    a[i] = a_block+i*NRA;

  for (i=0; i<NCA; i++)   /* Initialize pointers to b */
    b[i] = b_block+i*NCA;

  for (i=0; i<NRA; i++)   /* Initialize pointers to c */
    c[i] = c_block+i*NRA;

  for (i=0; i<NRA; i++)   /* Initialize pointers to res */
    res[i] = res_block+i*NRA;

    for (i=0; i<NRA; i++) /* last matrix has been initialized */
      for (j=0; j<NCA; j++)
        a[i][j]= (double) (i+j);
    for (i=0; i<NCA; i++)
      for (j=0; j<NCB; j++)
        b[i][j]= (double) (i*j);
    for (i=0; i<NRA; i++)
      for (j=0; j<NCB; j++)
        c[i][j]= 0.0;

    for (i=0; i<NRA; i++) {
      for(j=0; j<NCB; j++) {
        for (k=0; k<NCA; k++) {
          c[i][j] += a[i][k] * b[k][j];
        }
      }
    }
}

int main (int argc, char *argv[]) {

  int MyProc, tag=1, i, j;
  MPI_Request request;
  MPI_Status status;
  struct timespec req;
  int SIZE, IMBALANCE, ITERS;

  int flag=0;
  if (argc != 5)
  {
        req.tv_nsec = 5000;
        int SIZE = 100;
        int IMBALANCE = 10;
        int ITERS = 10;
  }
  else
  {
        req.tv_nsec = strtol(argv[1],NULL,0);
        SIZE = strtol(argv[2],NULL,0);
        IMBALANCE = strtol(argv[3],NULL,0);
        ITERS = strtol(argv[4],NULL,0);
  }
  //Aqui l'espera es de 5000 nanosegons, la idea seria jugar amb aquest valor
  //req.tv_nsec = 5000;
```

```
  req.tv_sec = 0;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &MyProc);

        if (MyProc == 0)
        {
  printf("nombre arguments %d\n", argc);
  printf("nsec %d\n", req.tv_nsec);
  printf("SIZE %d\n", SIZE);
  printf("IMBALANCE %d\n", IMBALANCE);
  printf("ITERS %d\n", ITERS);
        }


  for(i=0; i<ITERS; i++){
        for(j=0; j<IMBALANCE_BASE+IMBALANCE*MyProc; j++){
                matmul(SIZE);
        }

        MPI_Ibarrier(MPI_COMM_WORLD,&request);
        MPI_Test(&request, &flag, &status);
        while(!flag){
        clock_nanosleep(CLOCK, 0, &req, NULL);
        MPI_Test(&request, &flag, &status);
        }
        //MPI_Barrier(MPI_COMM_WORLD);


        if(MyProc == 0){
                printf("Iteration %d done!\n", i);
        }
  }

  MPI_Finalize();

  if(MyProc == 0){
        printf("Done.\n");
  }

  exit(0);
}
```

11.2 Càlcul del número π

11.2.1 Síncron

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
//
int main(int argc, char *argv[]) {

  int myid,nprocs;

  long int npts = strtol(argv[1],NULL,0);//1e10;

  long int i,mynpts;

  double f,sum,mysum;
  double xmin,xmax,x;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);

  if (myid == 0) {
    mynpts = npts - (nprocs-1)*(npts/nprocs);
  } else {
    mynpts = npts/nprocs;
```

```
  }

  mysum = 0.0;
  xmin = 0.0;
  xmax = 1.0;

  srand(myid);

  for (i=0; i<mynpts; i++) {
    x = (double) rand()/RAND_MAX*(xmax-xmin) + xmin;
    mysum += 4.0/(1.0 + x*x);
  }

  MPI_Reduce(&mysum,&sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

  if (myid == 0) {
    f = sum/npts;
    printf("PI calculated with %ld points = %f \n",npts,f);
  }
  MPI_Finalize();
}
```

11.2.2 Asíncron

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

#include <stdint.h>
//extra per asincron
#include <time.h>
#define CLOCK CLOCK_MONOTONIC

int main(int argc, char *argv[]) {

  int myid,nprocs;
  uint64_t npts;//1e10;

  int i;

  double f,sum,mysum;
  double xmin,xmax,x;

  int imbalance;
  //extra per asinron
  MPI_Request request;
  int flag=0;
  MPI_Status status;
  struct timespec req;
  req.tv_sec = 0;

  if (argc != 4)
  {//per defecte
    npts = 1000;
    req.tv_nsec = 5000;
        imbalance = 0;
  }
  else
  {
    npts = strtol(argv[1],NULL,0);//SIZE
    req.tv_nsec = strtol(argv[2],NULL,0);//temps espera
        imbalance = strtol(argv[3],NULL,0);//imbalance
  }

  int index,node,percent;
  uint64_t resta,aux;
  double imax,imin,jmax,jmin;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);

  uint64_t  mynpts[nprocs];
```

```
  if (myid == 0)
  {
        mynpts[0] = npts - (nprocs-1)*(npts/nprocs);
        aux = npts/nprocs;
        for (index=1;index<nprocs;index++)
        {
                mynpts[index]=aux;
        }
        srand(time(NULL));

        imin=0;
        imax=imbalance;
        jmin=0;
        jmax=nprocs-1;

        for (index=0;index<nprocs;index++)
        {
                percent = imin + (int)((double)rand() / RAND_MAX * (imax-imin));
                node = jmin + (int)((double)rand() / RAND_MAX * (jmax-jmin));
                resta = mynpts[index]*percent/100;
                mynpts[index] -= resta;
                mynpts[node] += resta;
                if (mynpts[node]<0)
                {
                        printf("Valor negatiu!");
                        exit(1);
                }
        }
   }

  MPI_Bcast(mynpts, nprocs, MPI_UINT64_T, 0, MPI_COMM_WORLD);

  mysum = 0.0;
  xmin = 0.0;
  xmax = 1.0;

  srand(myid);
  for (i=0; i<mynpts[myid]; i++) {
    x = (double) rand()/RAND_MAX*(xmax-xmin) + xmin;
    mysum += 4.0/(1.0 + x*x);
  }
  MPI_Ireduce(&mysum,&sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD,&request);
  MPI_Test(&request, &flag, &status);
  while (!flag)
  {
    clock_nanosleep(CLOCK, 0, &req, NULL);
    MPI_Test(&request, &flag, &status);
  }

  if (myid == 0) {
    f = sum/npts;
    printf("PI calculated with %ld points = %f \n",npts,f);
  }

  MPI_Finalize();
}
```