PRIMERA PART

Speed-up potencial

Abans de paral·lelitzar el codi proporcionat, ens interessarà coneixer la millora que obtindrem al aplicar la paral·lelització.
Aquesta mètrica la obtindrem al calcular l'speed-up mitjançant la llei d'Amdahl:

$$Speedup(n) = \frac{1}{F + \frac{1-F}{N}}$$

Dimensió matrius = 1000

Temps requerit per a executar la part paral·lela: 0,127267 segons.
Temps requerit per a executar tot el codi (mitjançant la comanda time): 0,402 segons

Percentatge del programa paral·lelitzable: 0,127267/0,402 * 100 = 31,66% Percentatge del programa no paral·lelitzable: 1- Part paral. = 68,34%

N. Threads	**2**	**4**	**8**	**16**	**32**	**64**	**128**
Speedup	1,188	1,3113	1,3831	1,42209	1,44239	1,45275	1,45799

Dimensió matrius = 1250

Temps requerit per a executar la part paral·lela: 0,270065 segons.
Temps requerit per a executar tot el codi (mitjançant la comanda time): 0,323 segons

Percentatge del programa paral·lelitzable: 0,270065/0,323 * 100 = 83'61%
Percentatge del programa no paral·lelitzable: 1- Part paral. = 16'38%

N. Threads	**2**	**4**	**8**	**16**	**32**	**64**	**128**
Speedup	1,71838	2,68158	3,72578	4,62657	5,26276	5,65132	5,86793

Dimensió matrius = 1500

Temps requerit per a executar la part paral·lela: 0,524078 segons.
Temps requerit per a executar tot el codi (mitjançant la comanda time): 0,608 segons

Percentatge del programa paral·lelitzable: 0.524078/0.608 * 100 = 86'2%

Percentatge del programa no paral·lelitzable: 1- Part paral. = 13’8%

N. Threads	2	4	8	16	32	64	128
Speedup	1,75742	2,82867	4,06874	5,21097	6,06184	6,60075	6,90780

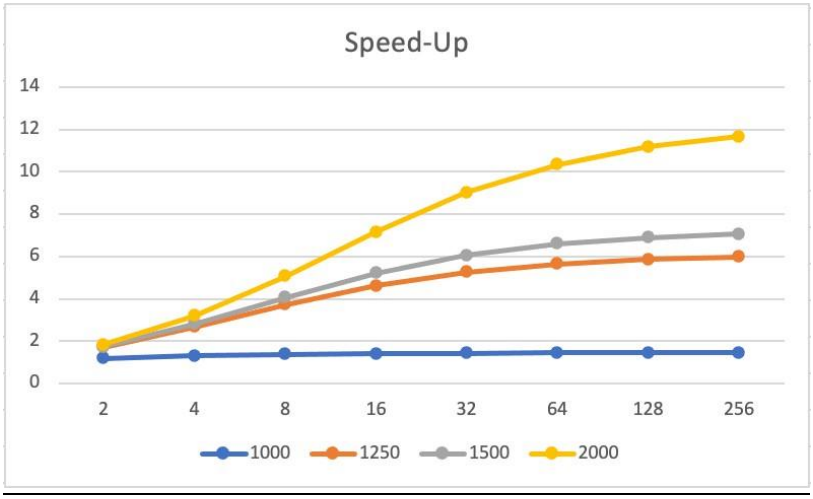
Dimensió matrius = 2000

Temps requerit per a executar la part paral·lela: 1.484962 segons.
Temps requerit per a executar tot el codi (mitjançant la comanda time): 1.618 segons

Percentatge del programa paral·lelitzable: 1.484962/1.618 * 100 = 91’77% Percentatge del programa no paral·lelitzable: 1- Part paral. = 8’23%

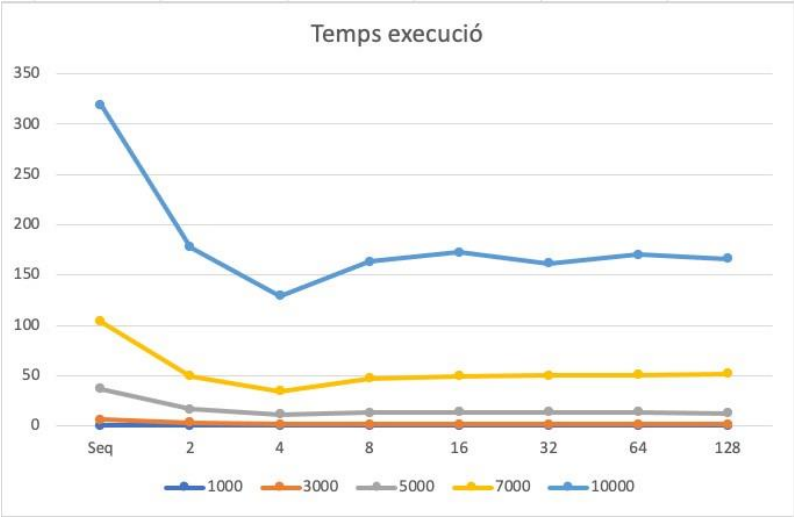
N.Threads	2	4	8	16	32	64	128
Speedup	1,84804	3,20854	5,07753	7,16410	9,01678	10,35582	11,1864506

Gràfic Speed-up



En el gràfic hem afegit també l’speed up al treballar amb 256 fils d’execució, ja que la corba de l’speed up al treballar amb matrius de dimensió 2000 no quedava plana del tot al arribar als 128 fils.
Com podem observar, paral·lelitzar el codi ens oferirà un speed-up de fins a 12 vegades respecte al codi seqüencial.

Temps d'execució de l'algorisme paral·lelitzat



Com podem observar en el gràfic, el temps d'execució del programa disminueix considerablement al passar de la versió seqüencial a la versió paral·lela amb dos fils. El temps d'execució més baix el trobem en la versió paral·lela amb quatre fils d'execució. A partir d'aquest punt, el temps s'estanca i inclús s'incrementa al treballar amb matrius de 7000 i 10000. Podem atribuir aquest fenomen a la saturació del processador al haver de treballar amb molts fils de manera paral·lela i concurrent.

SEGONA PART

Per a realitzar aquesta fase de la pràctica, primerament ens caldrà obtenir d'internet les característiques pròpies del processador Opteron 844, i així poder parametritzar el simulador Simplescalar de forma que es comporti de forma semblant a en aquest processador.

El processador serà CISC, volent dir això que executarà instruccions de tamany variable i de complexitat més elevada que les RISC, caracteritzades per la seva "simplicitat" envers les CISC i tamany fixe. Per a poder oferir un bon rendiment, les instruccions CISC que rebi el processador per executar seran traduïdes internament a altres instruccions RISC, requerint així d'unitats funcionals més simples.

Les seves característiques més rellevants seran:

L'Opteron 844 és un superescalar amb execució fora d'ordre 3-via, i va ser el primer AMD en fer el pas a l'arquitectura de 64 bits.

Referent a la k-via del processador, el Fetch i Decode són capaços d'obtenir de caché i descodificar fins a 3 instruccions per cada cicle.
Al no haver trobat a internet cap referència a la k-via de la fase de commit, assumirem que aquesta és també de 3.

```
# instruction fetch queue size (in insts)
-fetch:ifqsize                    3

# instruction decode B/W (insts/cycle)
-decode:width                     3

# instruction issue B/W (insts/cycle) -issue:width
11

# instruction issue B/W (insts/cycle)
-commit:width                      3
```

Un cop obtenides les microinstruccions al haver descodificat les macroinstruccions, aquestes seran enviades a les unitats funcionals o a la cua LSQ, en un màxim de fins a 11 microinstruccions per cicle.

Jerarquia de memòria

En quant a jerarquia de memòria, el processador disposarà dels següents nivells:

- L1 Instruction Caché: 64 KB, amb línies de 32 Bytes. Serà associativa a 2.
- L1 Data Caché: 64 KB, amb línies de 64 Bytes. Associativa a 2.
- L2 Caché: 1 MB, amb un tamany de línia de 64 Bytes. Associativa a 8.
- L3 Caché: 2 MB. Associativitat a 2.

Totes les caches seguiran la política de reemplaçament LRU.

S'usarà l'estàndard MOESI per a gestionar la coherència entre les caches.

```
#(tipus_cache,tamany_total,tamany_linia,associ,politica de reemplaza.)

# L1 inst cache config
-cache:il1              il1:64:32:2:l

# L1 data cache config
-cache:dl1              dl1:64:64:2:l

# L2 data cache config
-cache:dl2              ul2:1024:64:16:l

# L2 instruction cache config
-cache:il2                      dl2
```

Com que per internet no he trobat les latències de les diferents cachés que formen l'Opteron, amb microarquitectura K8, he decidit deixar les que Simplescalar tracta per defecte:

```
# L1 instruction cache hit latency  (in cycles)
-cache:il1lat                1

# L1 data cache hit latency  (in cycles)
-cache:dl1lat                1

# L2 data cache hit latency  (in cycles)
-cache:d21lat                6
```

Unitats Funcionals

El processador disposarà de 3 ALU's, 3 AGU's i 3 FPU's.
Això implicarà que el processador podrà executar de forma paral·lela (sempre que el Front End suministri prous instruccions) fins a 3 instruccions d'operacions d'enters (ALU), 3 d'operacions amb nombres reals (FPU) i 3 per a calcular direccions de memòria per a instruccions com Load i Store (AGU).

Per a Simplescalar, obviarem les AGU ja que el simulador no contempla la seva simulació.

```
# total number of integer ALU's available
-res:ialu                       3

# total number of integer multiplier/dividers available
-res:imult                      1

# total number of floating point ALU's available
-res:fpalu                      3
```

```
# total number of floating point multiplier/dividers available
-res:fpmult                          1
```

RUU i LSQ

Com podem apreciar al diagrama de blocs mostrat a continuació, cada nucli del processador podrà reordenar, mitjançant el Register Update Unit, fins a 72 instruccions (el tamany del RUU); i es podran gestionar fins a 44 instruccions de càrrega/descàrrega mitjançant la LSQ. Com que el simulador Simplescalar tant sols accepta valors potència de dos en aquests dos paràmetres, assignarem els valors exposats a continuació:

```
# register update unit (RUU) size
-ruu:size                           128

# load/store queue (LSQ) size
-lsq:size                           64
```

Ports de memòria + latències

En cada cicle, el processador serà capaç d'accedir a memòria dos cops, al disposar el sistema de dos ports de lectura/escriptura cap a memòria.

```
# memory access latency (<first_chunk> <inter_chunk>)
-mem:lat                54 3

# total number of memory system ports available (to CPU)
-res:memport                        2

# memory access bus width (en bytes, suposem 16 Bytes per cada port)
-mem:width                          16
```

Per a calcular el paràmetre first chunk (el temps en cicles que es tarda de mitjana al anar a buscar una dada de memòria i rebre la seva primera paraula), ens caldrà assumir que el processador estarà connectat a una memòria RAM compatible amb aquest, de la qual hem de conèixer la seva freqüència CAS i tRCD:

$$First\_chunk = CPU\_clock * (CAS+tRCD)/memory\_clock$$

$$Inter\_chunk = CPU\_clock / Memory\_Data\_Rate$$

Com que no he trobat cap memòria RAM DDR2 a internet on s'indiquessin les diferents latències (tant sols especifiquen la CAS), assumirem que les demés latències tindran el mateix valor que la CAS.

Mòdul RAM escollit
https://www.coolmod.com/v7-4gb-800mhz-pc2-6400-cl6-memoria-ddr2-sodimm-precio

FC = 1800*(6+6)/400 = 54 tics => anar a memòria a buscar una dada i retornar la primera paraula d'aquesta tardarà aquest nombre de tics de mitjana.
IC = 1800/800 = 2,25 = 3 tics => Un cop tinguem la primera paraula de la dada que ens interessa, els següents accessos a memòria per a agafar les següents paraules requeriran de 3 tics de rellotge cadascuna.

Predictor de salts

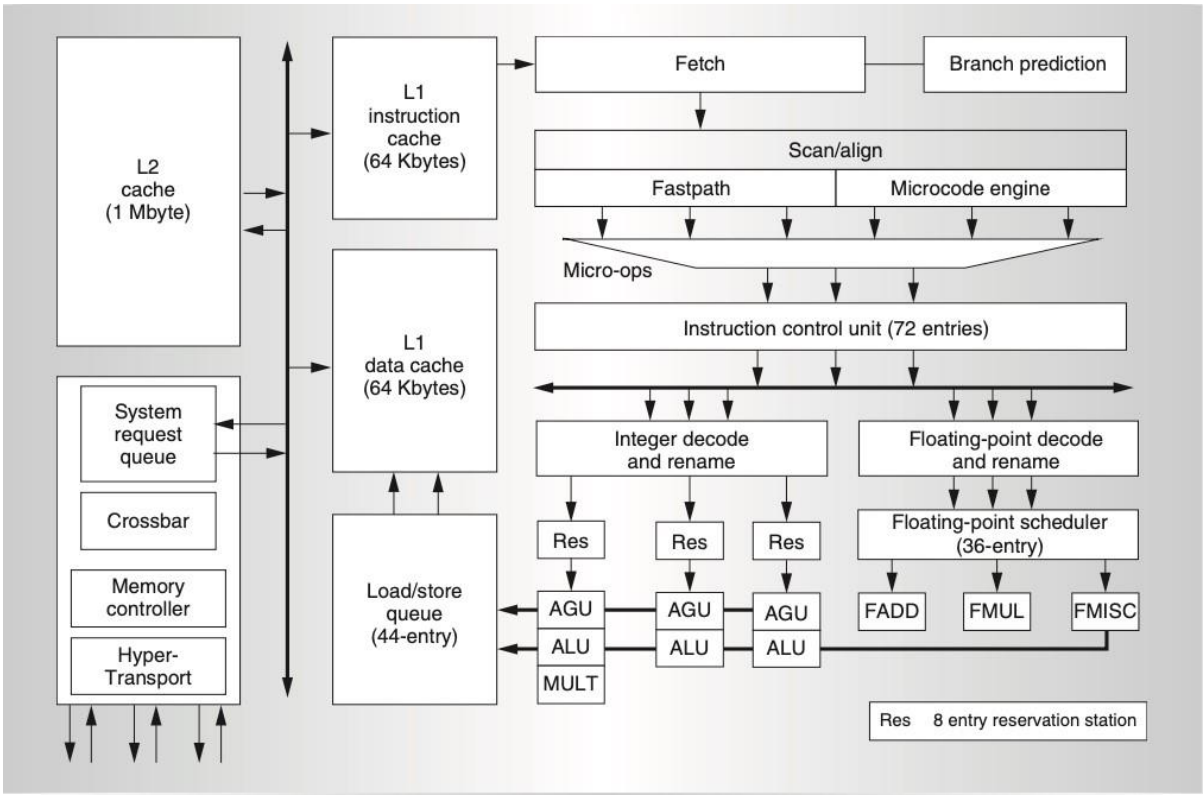
Com que no he trobat per internet el predictor de salts que usa el Opteron 844, he decidit usar el que possiblement més se li assemblaria, sent aquest un predictor de dos nivells (caracteritzant-se aquests per tenir estructures de dades auxiliars per a poder produir unes prediccions més acurades).

Usarem els valors per defecte de Simplescalar, i a continuació els millorarem per a veure com influeix en els resultats finals de la simulació.

```
# branch predictor type {nottaken|taken|perfect|bimod|2lev}
-bpred                          2lev

# 2-level predictor config (<l1size> <l2size> <hist_size>)
-bpred:2lev             1 1024 8
```

Diagrama de blocs del processador



Simulació amb els paràmetres Opteron bàsic

Un cop havent recopilat totes les dades referents al processador Opteron que volem simular, haurem d'executar un programa per veure com aquest rendeix.
El programa escollit serà el *MDxMD_M_alpha* proporcionat pels professors de la assignatura. El programa serà un multiplicador de matrius sequencial en format de binari alpha, compatible doncs amb Simplescalar.

Després d'executar el codi amb diferents tamanys de matrius, obtenim els següents resultats:

	1000	**1250**	**1500**	**2000**
Sim_IPC	2,2247	2,232	2,2374	1,7389
RUU_occupancy	118,3	119,14	119,74	124,04
LSQ_occupancy	29,66	29,63	29,62	31,02
2lev.bpred_dir_rate	0,995	0,9955	0,9959	0,9963
DL1_miss_rate	0,2574	0,2714	0,2825	0,2963
UL2_miss_rate	0,0015	0,009	0,0074	0,0466

Com podem veure en la taula, podem observar que **l'IPC** (nombre d'instruccions executades per cicle) **es manté constant fins que arribem a tractar amb matrius de 2000,** on empitjora. Intuitivament, podrem pensar que com major sigui la matriu, l'IPC serà pitjor.

Respecte a l'ús de la RUU, veiem que esta per sobre del 100% en tots els casos. Això vol dir que existirà una saturació en aquesta, que afectarà al rendiment general del sistema, fent fetch i decode de més instruccions de les que el RUU pot gestionar. Aquest serà un aspecte molt important a tractar respecte a les millores suggerides a continuació.
Analitzant ara el rendiment de **la cua LSQ**, veurem com esta es manté en un rang d'ús entorn al 30%, incrementant-se poc al passar a treballar amb matrius de 2000 * 2000 caselles.

En referència al **percentatge d'errors de la caché DL1**, podem veure com aquest es situa entre el 25-30%. Aquests valors cal reduir-los d'alguna forma, ja que una caché de dades de primer nivell amb un baix percentatge de falles és vital per a obtenir un rendiment òptim del sistema.

Finalment, veurem com **la caché L2 de dades** falla en molt poca proporció, arribant al 5% en les matrius de dimensió 2000.

Estudiant aquests resultats, veiem que incrementar el tamany de les cachés de primer nivell podria ser beneficiós per a reduir el percentatge d'errors, a costa d'obtenir latències majors.

La cua RUU ha de ser si o si ampliada, veient el grau de saturació que aquesta suporta, amb la conseqüent afectació al rendiment del sistema.

En referència al predictor de salts, veiem que el seu percentatger d'encerts es mostra molt alt, en part poden ser explicat per estar executant un codi amb uns patrons de salts bastant fàcils de trackejar, com seria un multiplicador de matrius. No contemplarem millorar el predictor de salts, al mostrar aquest un rendiment òptim.

Millores en el processador proposades

A continuació proposarem un seguit de millores a realitzar en el processador Opteron per a veure si influeixen en el rendiment general del processador.

En primer lloc, modificarem la K-VIA del processador, passant-la de 3 a 5. Farem aquest canvi perquè processadors més moderns en aquest disposen d'aquesta K-VIA, obtenint rendiments superiors, i per tant és un valor realista a provar. Per tant, ara en cada cicle es farà fetch, decode i commit de 5 instruccions/cicle. L'issue seguirà en 11 instruccions/segon.

```
# instruction fetch queue size (in insts) -fetch:ifqsize              4
# instruction decode B/W (insts/cycle) -decode:width                  4
 # instruction issue B/W (insts/cycle) -issue:width
11                                                                    4
# instruction issue B/W (insts/cycle) -commit:width                   4
```

Duplicarem també el nombre de ports a memòria disponibles per a realitzar lectures/escriptures, al treballar amb grans volums de dades com serien les matrius:

```
# total number of memory system ports available (to CPU)
-res:memport                    4
```

Duplicarem el tamany de la caché de dades (DL1) de primer nivell del sistema, ja que hem vist un percentatge d'error bastant alt en les simulacions (entorn al 25-30%).

```
# L1 data cache config
-cache:dl1            dl1:128:64:2:l
```

Com que el programa simulat requereix realitzar executar molts càlculs (al ser una multiplicadora de matrius), triplicarem el nombre d'unitats funcionals, tant d'enters com de coma flotant, dedicades a la multiplicació/divisió de nombres.

```
# total number of integer ALU's available
-res:ialu                       5

# total number of integer multiplier/dividers available
-res:imult                      3

# total number of floating point ALU's available
-res:fpalu                      5

# total number of floating point multiplier/dividers available
-res:fpmult                     3
```

A continuació, duplicarem el tamany de la cua RUU degut a la gran saturació que hem vist en aquest component durant les simulacions, i a la gran importància que aquest té sobre el rendiment general del processador.

```
# register update unit (RUU) size
-ruu:size                       256
```

Simulació amb els paràmetres Opteron millorat

Un cop ja hem preparat el fitxer de configuració de Simplescalar amb els paràmetres millorats del processador Opteron, passarem a repetir les simulacions per veure com el rendiment del processador a l'executar-les ha millorat/empitjorat.

	1000	**1250**	**1500**	**2000**
Sim_IPC	2,6974	2,3969	2,6918	2,0312
RUU_occupancy	213,9	215,0833	215,9230	227,4532
LSQ_occupancy	53,46	53,6716	53,8360	57,49
2lev.bpred_dir_rate	0,9941	0,9948	0,9952	0,9958
DL1_miss_rate	0,2296	0,2411	0,2516	0,2662
UL2_miss_rate	0,0127	0,0099	0,0081	0,0505

Com podem apreciar en la taula superior, l'IPC ha millorat en tots els casos, traduint-se en que els canvis que hem produït al processador han millorat el rendiment general del sistema. Veiem que el processador es beneficia d'una cua RUU més gran, mantenint-se la ocupació d'aquesta envoltant al 85%.
El predictor ha restat sense canvis, pero podem apreciar uns percentatges d'encert altíssims, superiors al 99%.
Per altra banda, veiem que la tassa de falles de la caché de dades de primer nivell (DL1) segueix elevada pràcticament a uns nivells semblants a la simulació prèvia a la millora, traduint-se en que podria fer falta inclus ampliar més el tamany d'aquesta. Tot i això, considerarem el nou tamany d'aquesta memòria caché vàlid, ja que seguir incrementant aquest possiblement abaixaria el percentatge de falles a costa d'incrementar notablement la latència d'accés en aquesta (aquest fenòmen no s'ha vist reflexat a Simplescalar, pel fet que en cap moment hem agrandat el valor de la latència d'aquesta).

Conclusions de la millora

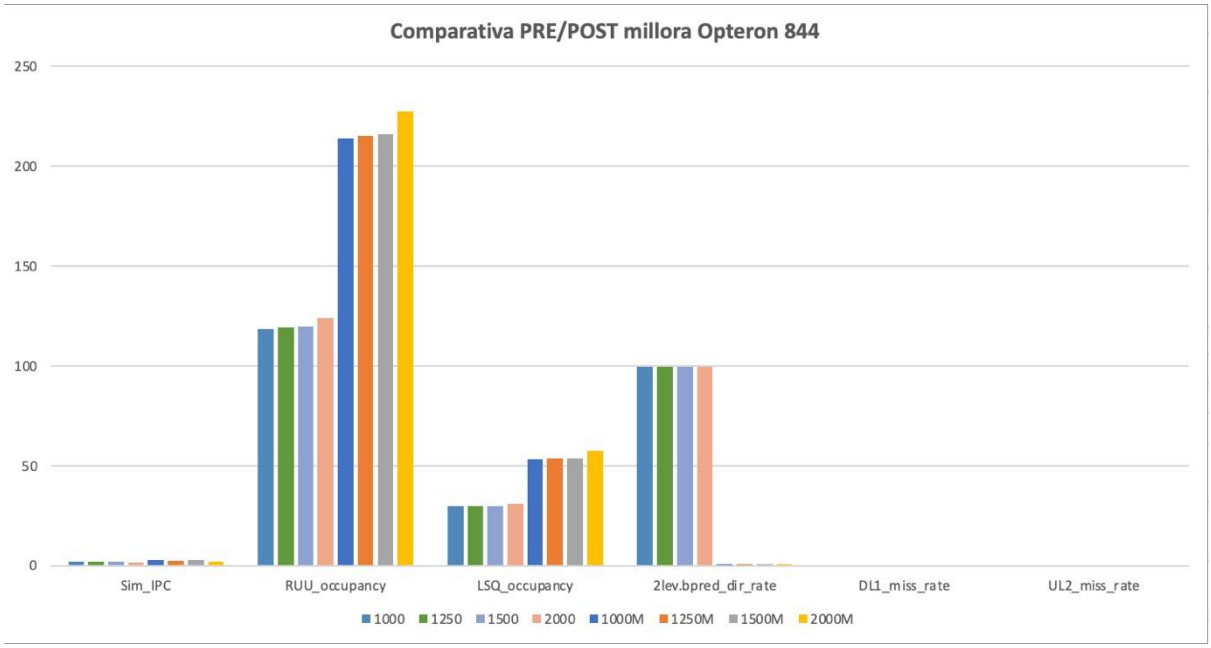
Veiem doncs en general una millora important, reflexada sobretot a l'IPC (que al cap i a la fi és la mètrica a la que més importancia li donarem).
El fet d'haver duplicat el nombre de ports d'accés a memòria també haurà provocat aquest increment del rendiment, al poder-se fer més accessos simultanis a memòria (a costa també d'empitjora la seva latència a nivell teòric, Simplescalar no ho té en compte).

Podriem dir que un dels fenomens decisius de millora ha estat una cua RUU més gran per encabir més instruccions executant-se simultaneament, ja que abans de la millora veiem un grau de saturació molt alt en aquest component del sistema.

Una RUU més gran també s'haurà vist beneficada d'una K-VIA del processador més ampla, capaç de nutrir-la amb més instruccions preparades per ser executades en cada cicle.

Veiem que la cua LSQ no es satura, tot i que el seu percentatge d'ús resulta ser bastant elevat (entorn al 82-85%).

Observem també que el fet d'haver incrementat el tamany de la DL1 no ha resultat tenir uns resultats com esperavem, tot i que de ben segur que augmentant-la més hauriem observat un menor miss rate.
Gràfic PRE/POST millora



Bibliografia

https://www.cpu-world.com/CPUs/K8/AMD-Opteron%20844%20%20OSA844CEP5AM.html

https://hardzone.es/2019/02/23/latencias-ram-mas-importantes-rendimiento-pc/

Apart de varis PDF's que he trobat per internet, com:

THE AMD OPTERON PROCESSOR FOR MULTIPROCESSOR SERVERS