CSEE W4824: Project Report

Nalini Vasudevan
Columbia University, NY
naliniv@cs.columbia.edu

Sambuddho Chakravarty
Columbia University, NY
sc2516@cs.columbia.edu

ABSTRACT

We present the design of our microprocessor for the Wall Street market which has been optimized for a given optimization function. The microprocessor has been optimized over two benchmarks, viz., blackscholes[1] and go [2] It may be implemented through a 32*nm* technology process with synchronous hardware design running at clock frequency of 5*GHz*.

The resulting microprocessor is a multicore processor with 13 cores of type *LargeCore* **having** 6*way* **issue units with a total area of approximately** 98mm^2**. The microprocessor is built in with on-core private Iinstruction Level-1 and Data Level-1 caches, each of size** 262144 **Kbytes,** [3]. The average miss-rate of the cores are approximately 0.05% for Data Level-1 caches and 0.00% for Instruction Level-1 caches when tested against blackscholes, as a best performance result. The average Cycles Per Instruction (CPI) in the best performance is about 0.76.**The optimal value of** *OF* **achievable by our microprocessor is** 46.33**.**

1. INTRODUCTION

The goal of this project is to design a microprocessor for a Wall Street company that uses two benchmarks [4], `go` [1] and `blackscholes` [3], each having 3 independent data sets.

Our motivation was to produce a design that gives considerable speed up, while keeping the total area of the microprocessor small.

The objective function to optimize is given by equation 1, where S={`blacksholes,go`},

D={`simsmall,simmid,simlarge`}.

To design our multi-core microprocessor we use the *sesc* [5] [4] simulator tool using a varied choice of configuration parameters given to us for obtaining the optimal *OF* parameter.

The problem doesn't seem likely to have any polynomial time solution [5], we attempt to interpret the simulation results/logs to determine which parameters to modify. The following parameters of the *sesc* simulator indicate the various factors of the program which indicate useful factors that show improvement in the Simulation Time (`sim_time`) reported by the *sesc*.

- *Cache MissRate* : Intuitive from the name, this parameter indicates the miss rate of the cache being used.
- *BJ* : Indicates percentage of instructions that correspond to branches and jumps.
- *Load/Store* : Indicates percentage of Load/Store (memory reference) instructions. This parameter is indicative of possible miss rate to expect and how much the improvement of cache type/size shall probably improve the `sim_time`.
- *INT/FP* : Indicates the percentage of integer and floating point instructions. The higher these are, the better seem chances of performance enhancement (provided the inter-dependence of instructions is less).

We followed a greedy approach for attaining our optimal configuration. Our focus was not towards optimizing die area used; our goal was to optimize `sim_time`. The *OF* factor is inversely proportional to the squareroot of the area of the die. *A total die area were* 100mm^2*, would affect the OF by only a factor of* 0.5..

Since *go* cannot be run on parallel threads of execution , we get the optimal performance by using the largest core available and the best cache size. We also observed that using specialized *IL*1 and *DL*1 the `sim_time` obtained were lower than that obtained with shared and/or unified cache. We used the cache size obtained from `go` for `blackscholes`. Our final configuration uses *LargeCores* cores for all cores to obtain lowest `sim_time` values. Since the *L*2 cache was required to be atleast twice as large as the *L*1 cache sizes, we decided not to use *L*2, and restricted the area of the die to be under 100mm^2.

We discuss in detail about our choices in the following section and report our results. The remainder of this report focuses on our design considerations , how we obtained the results and configurations we present and concludes with a brief note which summarizes all our findings and observations. our results.

$$OF = \frac{5}{\sqrt{OptimizedArea}} + \sum_{y \in D} \sum_{x \in S} \sqrt{\frac{baseSimTime\ of\ x\ with\ dataset\ y}{optimizedSimTime\ of\ x\ with\ dataset\ y}} \quad (1)$$

2. DESIGN CONSIDERATIONS

2.1 Cache Selection

Since *go* cannot be executed on multiple-threads, we started by testing the simulation with various cache configurations. We increased the cache size and associativity in so as to keep the access time small. With cache size greater than 10^5 bytes, the miss

[1]The blackscholes application calculates the prices of a portfolio of European options analytically with the Black-Scholes partial differential equation. For more information, visit http://en.wikipedia.org/wiki/Black-Scholes.

[2]This program was picked because the application expert believes that it can be used as a part of a new artificial intelligence package for financial applications. For further information on go, visit http://en.wikipedia.org/wiki/go

[3]1K = 1024

[4]Detailed description of the sesc simulator is beyond the scope of this document

[5]we have actually no way to verify that a polynomial time solution exits

rates were less than 1%. Figure 1 reports the configuration for Data Level 1 (*DL*1) for which the miss rates were very small. Similarly 2 reports the miss-rates for IL1. Since the miss-rate with 256*Kbytes* cache size was small in both cases, and the access time was 4 clock cycles, we decided to stick with this configuration for both *IL*1 and *DL*1.

Figure 3 gives the access time and area for the selected cache configuration. Figure 4 gives the number of cycles required to access the cache.

size	DL1 config	DL1 miss-rate
simsmall	size = 131072, assoc 8, blk size = 32	0.56%
simsmall	size = 262144, assoc 8, blk size = 32	0.01%

Figure 1: DL1 Cache miss rates for *go*

size	IL1 config	IL1 miss-rate
simsmall	size = 131072, assoc 8, blk size = 32	0.47%
simsmall	size = 262144, assoc 8, blk size = 32	0.02%

Figure 2: IL1 Cache miss rates for *go*

Blackscholes can be executed through simultaneous parallel threads of execution. Each of these threads can execute on a separate processing core. So, for speeding up this program, we decided to run it on separate cores. This subsection and the next one focuses on how the cache size and processing core was selected for *blackscholes*.

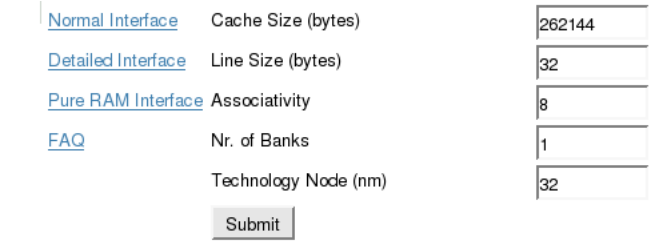
Initially, we noticed that with the default configuration UL1S.conf, when executing the `smallsim` input data set using unified shared cache -32Kbytes/32byte(block size)/2 way-set associative) resulted in high miss rate (close to 70%). Thus we decide to increase the cache size a bit and added an Level 2 (*L*2) cache as well. No improvement in simulation time parameter (`sim_time`) of the sesc simulator was observed. Thus, we decided to add a specialized Instruction and Data Level 1 (*IL*1 and *DL*1)(IL1DL1P.conf configuration) cache to the processor. This drastically improved the `sim_time` to 292.47*msec*. Improving the cache associativity from 2-way to 4-way further improves the `sim_time` to 266.49*msec*. The *L*2 cache misses were still approximately 30.62%. Increasing the *L*2 cache size to 64Kbytes resulted in only slight improvement of the performance (`sim_time` of 250.55*msec*). However, the *L*2 misses drastically decreased to 4.33%.

Further improvement was contingent on the better selection of multiple processing cores. There are approximately 30% integer operations. The more the number instructions fetched per clock cycles, higher are the chances of fetching the integer and floating point operations. Moreover the LargeCore CPUs have larger number of parallel execution units. Thus, further improvement of the `sim_time` was observed with more processing cores. The intuition of behind seeking better `sim_time` is that it has a greater impact on the *OF* parameter (than area, which only affects the OF by an amount of 0.5).

2.1.1 Core Selection

As we just mentioned in the previous subsection, better `sim_time` was achieve with higher core selection with larger number of instructions fetched per clock cycles. Thus we tried executing the `smallsim` data set using multiple core processors with higher number of instructions fetched.

Figure 5 borrowed from [4], shows the effect of number of cores on the benchmarks. Therefore at around $n = 8$ ideal speed up is obtained. This gave us some intuition that the optimal performance



Cache Parameters:

Number of banks:1
Total Cache Size (bytes):262144
Size in bytes of bank:262144
Number of sets per bank:1024
Associativity:8
Block Size (bytes):32
Read/Write Ports per bank:1
Read Ports per bank:0
Write Ports per bank:0
Technology Size (nm):32
Vdd:0.9

Access time (ns): 0.76464237931
Random cycle time (ns):0.311804261729
Multisubbank interleave cycle time (of data array) (ns):0.26083536566
Total read dynamic energy per read port(nJ): 0.0966943932482
Total read dynamic power per read port at max freq (W): 0.310112481183
Total standby leakage power per bank (W): 0.0631519414866
Refresh power (percentage of standby leakage power): 0.0
Total area (mm^2): 0.78420276383
DRAM array refresh interval (microseconds):0.0

Figure 3: Output of *cacti* for our cache configuration

$$\begin{aligned} \text{Cycles for Cache Access} &= \text{[Access Time * Clock Frequency]} \\ &= [0.7647 * 10^{-9} * 5 * 10^{9}] \\ &= [3.8235] = 4 \end{aligned}$$

Figure 4: Access time for cache

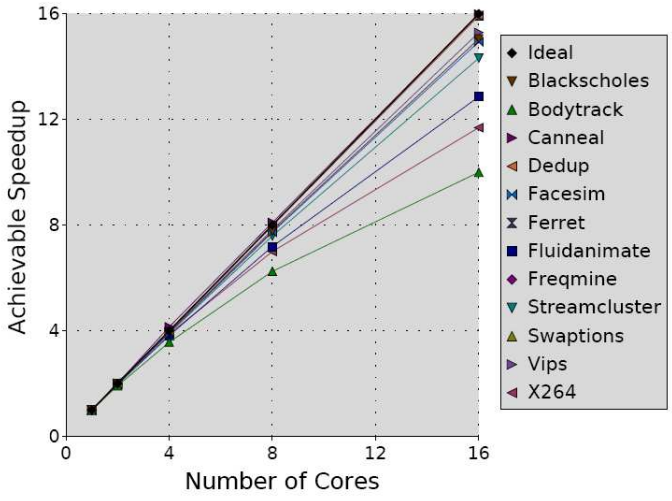


Figure 5: Effect of cores on the benchmarks (Borrowed from [4]).

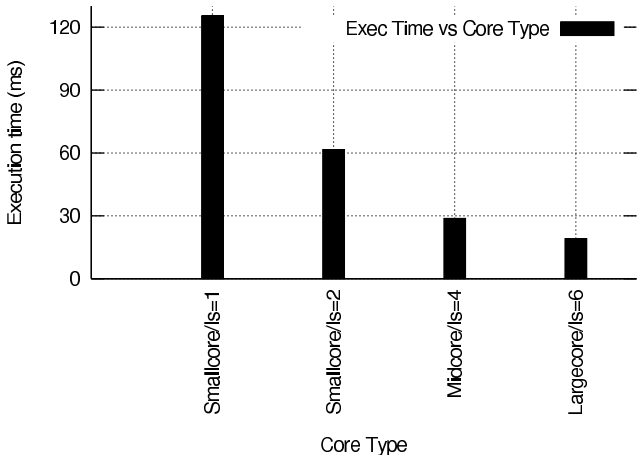


Figure 6: Variation of time with the type of core for *blackscholes*.

should be around 8 processors. This fact further reflects in the experimental observations presented in this subsection.

For sake of brevity we do not describe the results from all the core types that we tested and what results we saw; but instead only present a summary of some of crucial improvements. The following table list some of the key improvements we observed for selection of the core types and quantities and appropriate cache quantities. Tables 1, 2 and 3 list the CPU core selection with cache configuration. Table 1 lists cases with decreasing `sim_time`. Smaller the values of `sim_time` gives better values of our optimization function *OF*. `sim_time` has a greater effect on the *OF*. [6]

The tables list core and cache configurations which demonstrate better `sim_time` values. The names of the parameter and the values in all three of the tables are intuitive. Core type column indicate some of the types of cores against which *blackscholes* was tested. For each of the values of the core types, the name of the core is followed by the number of issue units in each of the cores. In case of *SmallCore* type of cores, the ones with issue width of 1 are the one which have in-order execution. The rest all have out-of-order execution units.

Cache Config column describes the names/types of cache configurations used from [2]. Rest of the fields are intuitive and have been directly selected from the `sesc` simulator configuration parameters.

LargeCore definitely gives higher performance over other other core types and this is demonstrated in Figure 6. Since we did not care about area, we chose large cores for all. We also tried several hybrid combinations of *LargeCores* with *MidCores*, but the performance was suboptimal compared to that using only *LargeCores*

Figure 7 shows the variation of `sim_time` with the number of cores. We used *LargeCore* core types for all cores; the total die area was restricted to $100mm^2$. We knew the size of the $IL1/DL1$ caches. This allowed us to use atmost 13 cores; with 14 cores the area exceeded $100mm^2$.

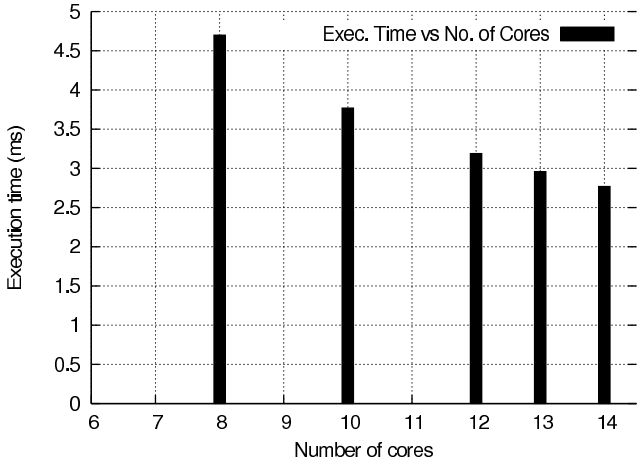


Figure 7: Variation of `sim_time` with the number of cores with our experiments for *blackscholes*.

[6] Using up the entire die area of $100mm^2$ would only affect on the value of *OF* by a factor of 0.5.

# of Cores	Core Type	Area/ Core (mm^2)	Cache Type	Cache Config	Area Used by Cache (mm^2)	Total Area (mm^2)	Sim Time *msec*.	Base Sim Time *msec*.
1	SmallCore/1	0.05	UL1S	32K/8B/2way	0.117	0.17	424.77	424.77
1	SmallCore/1	0.05	IL1DL1S_L2 64K/32B/4way($L2$)	32K/32B/2way($IL1/DL1$)	0.44	0.49	255.57	424.77
2	SmallCore/1	0.05	IL1DL1P_L2 64K/32B/4way($L2$)	32K/32B/2way($IL1/DL1$)	0.674	0.87	125.45	424.77
4	SmallCore/2	0.1	IL1DL1P_L2	32K/32B/2way($IL1/DL1$) 64K/32B/4way($L2$)	1.14	1.54	61.63	424.77
8	SmallCore/2	0.1	IL1DL1P_L2	32K/32B/2way($IL1/DL1$) 64K/32B/4way($L2$)	2.08	2.88	31.63	424.77
12	SmallCore/2	0.1	IL1DL1P_L2	32K/32B/2way($IL1/DL1$) 64K/32B/4way($L2$)	3.01	4.21	21.78	424.77
16	SmallCore/2	0.1	IL1DL1P_L2	32K/32B/2way($IL1/DL1$) 64K/32B/4way($L2$)	3.94	5.54	16.86	424.77
4	MidCore/4	2.11	IL1DL1S	128K/32B/8way($IL1/DL1$)	1.02	9.46	14.72	424.77
8	MidCore/4	2.11	IL1DL1S	256K/32B/8way($IL1/DL1$)	1.57	18.45	12.54	424.77
12	MidCore/4	2.11	IL1DL1S	512K/32B/8way($IL1/DL1$)	2.65	27.97	11.82	424.77
4	LargeCore/6	5.92	IL1DL1S	128K/32B/8way($IL1/DL1$)	1.02	24.70	11.21	424.77
8	LargeCore/6	5.92	IL1DL1S	256K/32B/8way($IL1/DL1$)	1.57	48.93	11.15	424.77
13	**LargeCore/6**	**5.92**	**IL1DL1P**	**256K/32B/8way($IL1/DL1$)**	**2.04**	**97.35**	**2.96**	**424.77**

Table 1: Some Key Results of Core Selection and Cache Configuration for BlackScholes using `SmallSim` data-set

# of Cores	Core Type	Area/ Core (mm^2)	Cache Type	Cache Config	Area Used by Cache (mm^2)	Total Area (mm^2)	Sim Time *msec*.	Base Sim Time *msec*.
4	SmallCore/2	0.1	IL1DL1S	128K/32B/8way($IL1/DL1$)	1.02	1.42	44.34	865.56
8	SmallCore/2	0.1	IL1DL1S	256K/32B/8way($IL1/DL1$)	1.57	2.37	30.47	865.56
12	SmallCore/2	0.1	IL1DL1S	512K/32B/8way($IL1/DL1$)	2.65	3.85	25.1	865.56
16	SmallCore/2	0.1	IL1DL1S	512K/32B/8way($IL1/DL1$)	2.65	4.25	25.63	865.56
4	MidCore/4	2.11	IL1DL1S	128K/32B/8way($IL1/DL1$)	1.02	9.46	30.24	865.56
8	MidCore/4	2.11	IL1DL1S	256K/32B/8way($IL1/DL1$)	1.57	18.45	25.00	865.56
12	MidCore/4	2.11	IL1DL1S	512K/32B/8way($IL1/DL1$)	2.65	27.97	23.56	865.56
16	MidCore/4	2.11	IL1DL1S	512K/32B/8way($IL1/DL1$)	2.65	36.41	23.84	865.56
4	LargeCore/6	5.92	IL1DL1S	128K/32B/8way($IL1/DL1$)	1.02	24.70	30.24	865.56
8	LargeCore/6	5.92	IL1DL1S	256K/32B/8way($IL1/DL1$)	1.57	48.93	22.31	865.56
13	**LargeCore/6**	**5.92**	**IL1DL1P**	**256K/32B/8way($IL1/DL1$)**	**2.04**	**97.35**	**5.92**	**865.56**

Table 2: Some Key Results of Core Selection and Cache Configuration for BlackScholes Using `MidSim` data-set

# of Cores	Core Type	Area/ Core (mm^2)	Cache Type	Cache Config	Area Used by Cache (mm^2)	Total Area (mm^2)	Sim Time *msec*.	Base Sim Time *msec*.
4	SmallCore/2	0.1	IL1DL1S	128K/32B/8way($IL1/DL1$)	1.02	1.42	537.00	3472.47
8	SmallCore/2	0.1	IL1DL1S	256K/32B/8way($IL1/DL1$)	1.57	2.37	319.89	3472.47
12	SmallCore/2	0.1	IL1DL1S	512K/32B/8way($IL1/DL1$)	2.65	3.85	188.77	3472.47
16	SmallCore/2	0.1	IL1DL1S	512K/32B/8way($IL1/DL1$)	2.65	4.25	170.60	3472.47
4	MidCore/4	2.11	IL1DL1S	128K/32B/8way($IL1/DL1$)	1.02	9.46	127.15	3472.47
8	MidCore/4	2.11	IL1DL1S	256K/32B/8way($IL1/DL1$)	1.57	18.45	100.48	3472.47
12	MidCore/4	2.11	IL1DL1S	512K/32B/8way($IL1/DL1$)	2.65	27.97	96.12	3472.47
16	MidCore/4	2.11	IL1DL1S	512K/32B/8way($IL1/DL1$)	2.65	36.97	96.71	3472.47
4	LargeCore/6	5.92	IL1DL1S	128K/32B/8way($IL1/DL1$)	1.02	24.70	127.15	3472.47
8	LargeCore/6	5.92	IL1DL1S	256K/32B/8way($IL1/DL1$)	1.57	48.93	89.19	3472.47
13	**LargeCore/6**	**5.92**	**IL1DL1P**	**256K/32B/8way($IL1/DL1$)**	**2.04**	**97.35**	**23.65**	**3472.47**

Table 3: Some Key Results of Core Selection and Cache Configuration for BlackScholes Using `LargeSim` data-set

2.2 TLB Selection

Increasing the TLB for both *go* and *blackscholes* benchmarks resulted in no improvement of the `sim_time`, and therefore we decided not have a TLB. This is summarized in the table in Figure 8.

Benchmark	data-set	Core Type	TLB	`sim_time` (*ms*.)
blackscholes	`simsmall`	4 SmallCores	No	63.4
blackscholes	`simsmall`	4 SmallCores	Yes	63.4
go	`simsmall`	1 LargeCore	No	45.981
go	`simsmall`	1 LargeCore	Yes	45.981

Figure 8: Experiments to test the usage of TLB (TLB configuration: 2048 bytes, 2 way associativity, 32 byte line size)

Evident from Figure 8, we see no improvement with addition of TLB to the simulator, for neither *blackscholes* nor *go* benchmarks. Hence we decided not to use the TLB [7].

From these results we concluded that using 13 *LargeCore* cores with 6 issue units per core and using on-core specialized caches, each of size 256Kbytes (262144 Bytes), would have the optimum effect on *OF* due to *blackscholes*. The Same configuration may be used for *go* which is cannot be run with separate threads of execution. We selected *go* and ran it on one of the cores and obtained the results presented in figure 10. Evident from the result (and also based on our previous knowledge of *go*), it mostly requires higher number of execution units (due to high percentage of integer and floating point units).

3. FINAL RESULTS AND DESIGN

This section summarizes our optimal results. The table in figure 9 and 10 presents the values of the base `sim_time`.

BaseSimTime(*ms*)	`simsmall`	`simmid`	`simlarge`
blackscholes	424.766	867.560	3472.466
go	458.139	1172.827	2986.398

Figure 9: Base SimTime

Optimized SimTime(*ms*)	`simsmall`	`simmid`	`simlarge`
blackscholes	2.958	5.912	23.657
go	45.888	113.247	283.871

Figure 10: Optimized SimTime

Using the *LargeCore* cores selected and the cache configuration, the total die area is calculated to be $97.3518 mm^2$. The equations in figure 3 summarize the details of this calculation.

The base times and the optimized times are shown in Figure 9 and 10 respectively. The calculation of area is shown in Figure 3. We solve Equation 1 with optimized values from Figure 10 in Figure 3.

Thus, by solving the equation above in Figure 3 we get a final *OF* of **46.33**. *Thus using 13 LargeCore cores, specialized IL*1/*DL*1 *cache of* 256*Kbytes per core (having* 32*Byte blocks and* $8-way$ *associativity), we attain an optimal OF of* 46.33.

[7] though the area expended due to addition of TLB has negligible effect on the total die area

Core Area	=	13 * Area(Each Core) = 13 * 5.92	=	76.96)
IL1 Area	=	13 * Area(Each IL1Cache) = 13 * 0.7843	=	10.1959
DL1 Area	=	13 * Area(Each DL1Cache) = 13 * 0.7843	=	10.1959
L2 Area	=	0 (No L2)	=	0
		Total Area (mm^2)	=	**97.3518**

Figure 11: Area Calculation

$$OF = \frac{5}{\sqrt{OptimizedArea}} + \sum_{y \in D} \sum_{x \in S} \sqrt{\frac{baseSimTime\ of\ x\ with\ dataset\ y}{optimizedSimTime\ of\ x\ with\ dataset\ y}}$$

$$= \frac{5}{\sqrt{97.3518}} + (\sqrt{\frac{424.766}{2.958}} + \sqrt{\frac{867.560}{5.912}} +$$

$$\sqrt{\frac{3472.466}{23.657}} + \sqrt{\frac{458.139}{45.888}} + \sqrt{\frac{1172.827}{113.247}} + \sqrt{\frac{2986.398}{283.871}})$$

$$=0.51+(11.98+12.11+12.11+3.16+3.22+3.24)$$

$$=\mathbf{46.33}$$

Figure 12: Solving for *OF*

3.1 Final Design

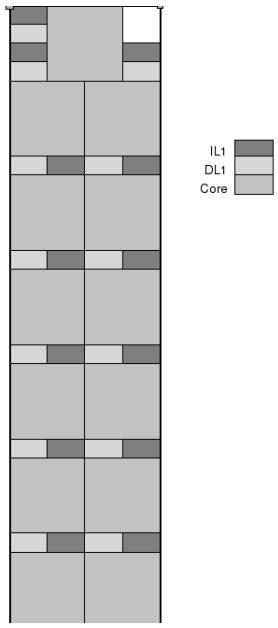


Figure 14: Approximate floor plan of our microprocessor (Buses not shown).

The block diagram of our processor is as shown in figure 3.1. It shows 13 cores (each of *LargeCore* with issue width of 6 instructions). Each core has separate specialized *IL*1 and *DL*1 cache. We used no L2 cache. The *IL*1 and *DL*1 are each of size 256*Kbytes* (262144*bytes*). The caches have 32*byte* block size with $8-way$ set associativity. We have derived this through a series of experimentation and inferences based on how the `sim_time`, cache misses and CPI is improved when various simulation configuration param-

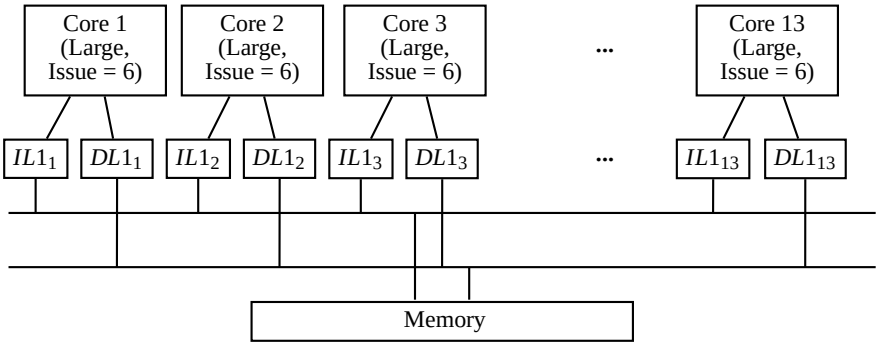


Figure 13: Our final design of the microprocessor. Each IL1 and DL1 has the following configuration: 262144B cache, 32B line size, 8 way associativity.

eters are varied. Figure 14 shows the approximate floor plan of the processor. It shows the approximate physical layout of the cores and their caches on a $100mm^2$ die. The communication buses between the cores is not being shown here as they have already been taken care of by the simulator.

4. CONCLUSIONS AND REMARKS

From the results section, it can be seen that the speed-up we achieved for *blackscholes* is far greater than that of *go*. This is because *blackscholes* can be parallel threads of simultaneous execution, but the speed up obtained for *go* is accounted only from the increased cache size and the processor type.

The area factor in the formula for calculating *OF* is deceiving: Assume, $f_1 = \frac{5}{\sqrt{OptimizedArea}}$.

Suppose the Optimized Area is $25mm^2$, then this factor would evaluate to $5/\sqrt{25}$ $(= 1)$. On the other hand if the optimized area were $100mm^2$ (the maximum possible value), the factor would evaluated to $5/\sqrt{100}$ $(= 0.5)$. So with this calculation, we see that increasing the are from $25mm^2$ to $100mm^2$ only reduces the OF by 0.5. Therefore the first factor (f_1), containing area really does a relatively large impact on the *OF* factor. Consequently it does substantially help including the area in the formula for *OF*. *Hence we did not try to optimize area*.

For this reason, we suggest two possibly new optimization functions: Since we are restricted to using $100mm^2$, we remove the area factor from the formula of *OF* altogether.

$$OF = \sum_{y \in D} \sum_{x \in S} \sqrt{\frac{baseSimTime\ of\ x\ with\ data-set\ y}{optimizedSimTime\ of\ x\ with\ dataset\ y}} \quad (2)$$

With this metric, the OF evaluates to 45.82 for our configuration.

The second metric we suggest is to cause the area to contribute to a greater extent to the *OF*.

$$OF = \frac{1}{\sqrt[k]{OptimizedArea}} \times \sum_{y \in D} \sum_{x \in S} \sqrt{\frac{baseSimTime\ of\ x\ with\ dataset\ y}{optimizedSimTime\ of\ x\ with\ dataset\ y}} \quad (3)$$

In the above equation, the area factor is multiplied by the time factor, therefore the area having a greater influence on the calculation of *OF*. k can be adjusted depending on the amount of impact of area wanted on *OF*. Less the value of *k*, greater is the impact. Suggested value of k is 2 or 3.

Overall, we enjoyed doing the project. It was a great learning experience. Initially, we thought that the project would take a large number of simulations to come to conclusions, but in reality it does not.

5. REFERENCES

[1] Blackscholes. `http://en.wikipedia.org/wiki/Go\_(board\_game)`.

[2] Csee w484: Course project description. http://www1.cs.columbia.edu/ cs4824/handouts/project.pdf.

[3] Go (game). `http://en.wikipedia.org/wiki/Black-Scholes`.

[4] BIENIA, C., KUMAR, S., SINGH, J. P., AND LI, K. The parsec benchmark suite: characterization and architectural implications. In *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques* (New York, NY, USA, 2008), ACM, pp. 72–81.

[5] SESC: SuperESCalar Simulator. `http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/`.

Acknowledgment

We would like to thank Prof. Luca Carloni and the TAs (Young Jin Yoon, Bharadwaj Vellore and Rebecca Collins) for their valuable suggestions and feedback throughout the course of this project. Thanks to Young Jin for designing an interesting project.

APPENDIX

A. CONFIG FILE (*ILDL1P.CONF*)

```
\texttt{
# You can start modifying from the line below.
#####################################################

#1. multicore configuration.
procsPerNode = 13 # total number of cores.
cpucore[0:12] = 'LargeCore'

#2. Issue width for each core types (i.e. small, mid, and large.)
issueLarge = 6 # large-processor issue width (6 or 4)
issueMid = 4 # mid-processor issue width (4 or 2)
issueSmall = 1 # small-processor issue width (2 or 1)

#3. IL1 & DL1 configuration for large cores
LargeIL1CacheSize = 262144
LargeIL1BlockSize = 32
LargeIL1Assoc = 8
LargeIL1AccessTime = 4 # Need to be calculated by CACTI

LargeDL1CacheSize = 262144
LargeDL1BlockSize = 32
LargeDL1Assoc = 8
LargeDL1AccessTime = 4 # Need to be calculated by CACTI

#4. IL1 & DL1 configuration for mid cores
MidIL1CacheSize = 32768
MidIL1BlockSize = 32
MidIL1Assoc = 2
MidIL1AccessTime = 3 # Need to be calculated by CACTI

MidDL1CacheSize = 32768
MidDL1BlockSize = 32
MidDL1Assoc = 2
MidDL1AccessTime = 3 # Need to be calculated by CACTI

#5. IL1 & DL1 configuration for small cores
SmallIL1CacheSize = 32768
SmallIL1BlockSize = 32
SmallIL1Assoc = 2
SmallIL1AccessTime = 3 # Need to be calculated by CACTI

SmallDL1CacheSize = 32768
SmallDL1BlockSize = 32
SmallDL1Assoc = 2
SmallDL1AccessTime = 3 # Need to be calculated by CACTI

#6. SmallCore inorder/out-of-order configuration
[SmallCore]
inorder = true # does the core execute in order?

#7. Translation Lookaside Buffer(TLB) for data & inst. addresses in LargeCore.
[FXDTLBLarge]
size = 64*8
assoc = 4
bsize = 8
deviceType = 'none'

[FXITLBLarge]
size = 64*8
assoc = 4
bsize = 8
deviceType = 'none'

#8. Translation Lookaside Buffer(TLB) for data & inst. addresses in MidCore.
[FXDTLBMid]
size = 64*8
assoc = 4
bsize = 8
deviceType = 'none'

[FXITLBMid]
size = 64*8
assoc = 4
bsize = 8
deviceType = 'none'

#9. Translation Lookaside Buffer(TLB) for data & inst. addresses in SmallCore.
[FXDTLBSmall]
size = 64*8
assoc = 4
bsize = 8
deviceType = 'none'

[FXITLBSmall]
size = 64*8
assoc = 4
bsize = 8
deviceType = 'none'

# Please do not modify below this line
#####################################################
}
```