# Compiling Issues for the Simultaneous Multithreading Processor

Diego Puppin (diego.puppin@sns.it)

Scuola Normale Superiore

Università degli Studi di Pisa

University of California at San Diego

# Agenda

① Simultaneous multithreading

② Parallel templates for SMT

③ Performance model

④ Conclusion and future work

⑤ Further information

## Introduction

**Goal:** understanding of what a smart compiler can do to exploit high performance out of the SMT processor;

**Methodology:** two different issues explored:

1. standard techniques (loop fusion, loop peeling, cyclic reduction...) to *parallelize* the kernels; multithreaded versions' performance measured and discussed;
2. performance model based on compile-time information (basic blocks' size, critical path, loop-critical path); model for processor utilization and the best number of thread developed and evaluated.

## Problems with processor design

How to get proportional performance out of the available chip area?

- add more memory (huge caches, up to 94% chip area)
- increase the system integration (controllers on chip)
- new design approachs (RAW, PIM, IRAM)
- *increase available parallelism*

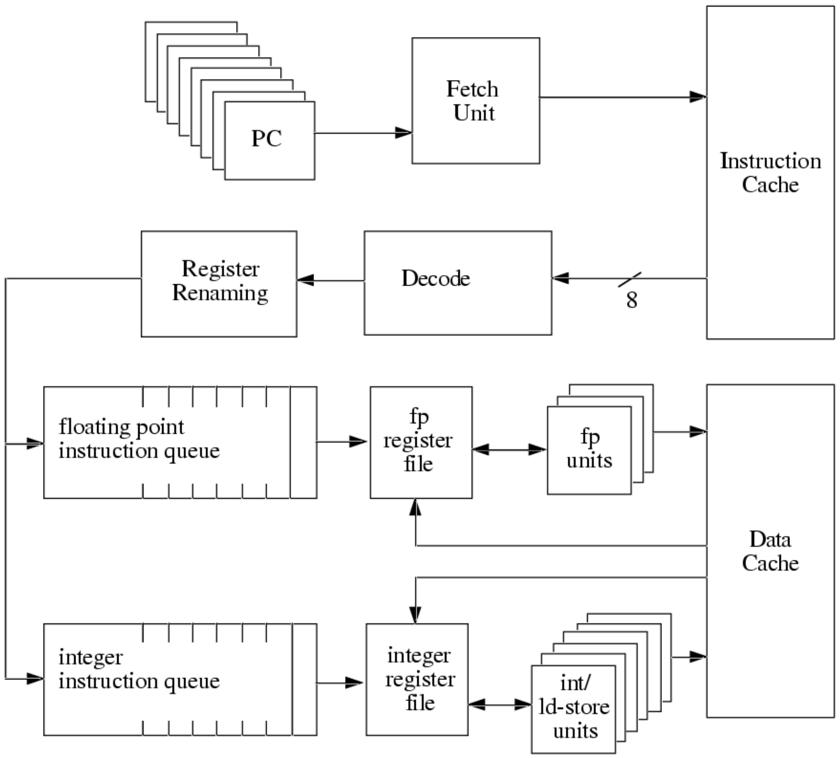
## How to increase available parallelism

Present day processors features various paradigms of parallelism, among which pipeline, functional units replication, specialization, multithreading (MT):

- coarse-grained MT: MIT Alewife,
- fine-grained MT: HEP, Cray MTA,
- simultaneous MT: SMT.

Simultaneous multithreading is “thread-level as instruction-level parallelism”: the ability to issue and execute instructions from different threads *in the same cycle*.

# From superscalar to SMT



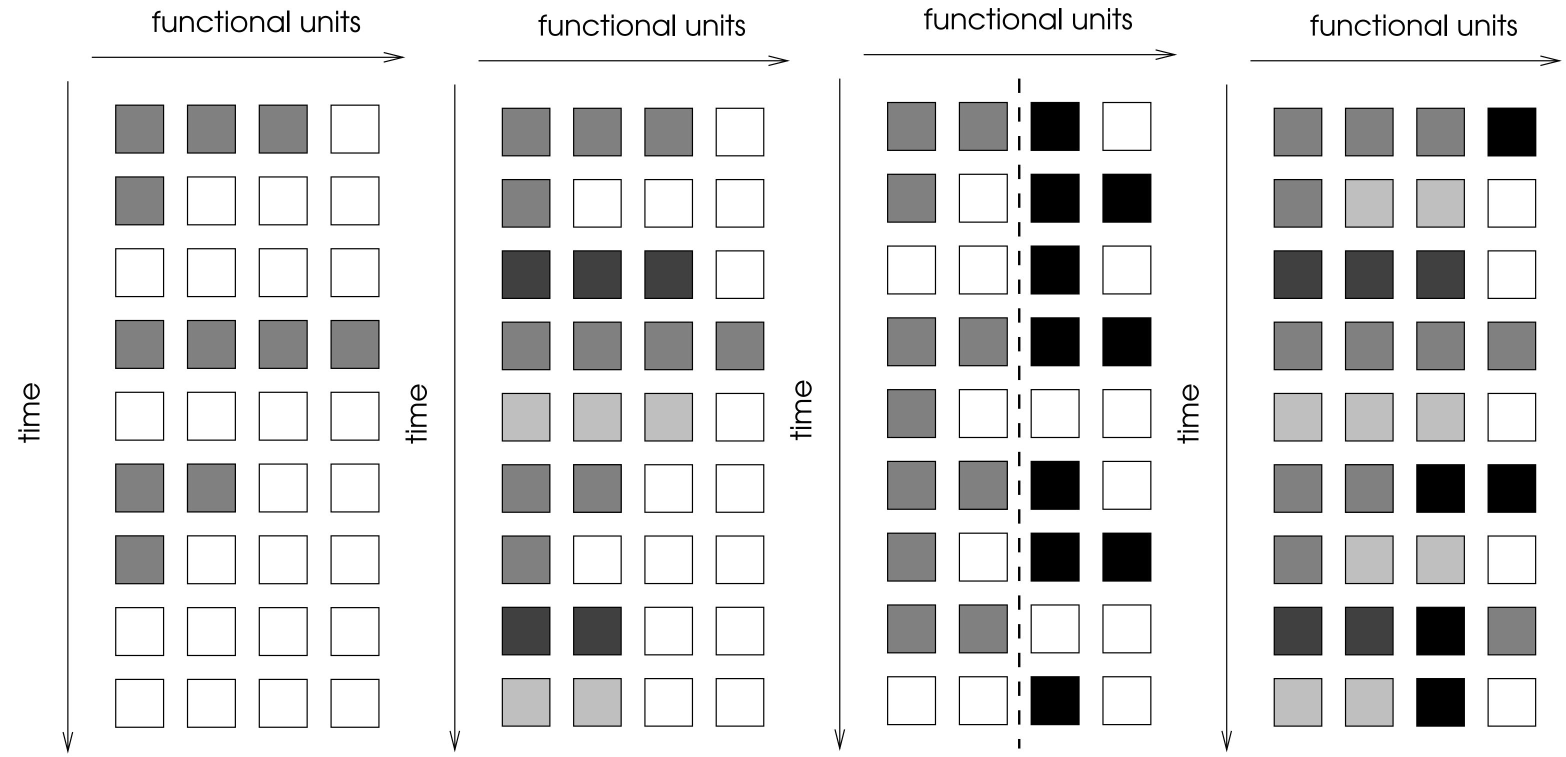

Basic SMT architecture (from MIPS R10000)

Added:

- multiple PC,
- independent return stack, instruction retirement, instruction queue flush and trap mechanisms for each thread,
- tagged entries in BTB,
- larger register file .

No need for independent instruction queues or functional units after register renaming!

# Simultaneous multithreading: main idea




Superscalar, fine-grained multith., chip-multiproc. and SMT

John Hennessy, in his speech at Microprocessor Forum 1999, showed that 4-wide superscalar processors rarely achieve a sustained a 2 instruction-per-cycle rate under *Spec95* benchmarks (**?**).

Fine-grain multithreaded processors, able to change context every cicle without degradation, seem to be a good approach to this problem, as they are able to interleave the execution of different threads in order to hide dependencies and latencies. Some studies anyway hold that they are not able to utilize more than 40% of a wide superscalar execution bandwidth (**?**). It should be clear, however, that even if vertical wastes are avoided by multithreading, a single thread may not be able to fill the whole execution bandwidth, due to the limited ILP it can offer. This problem is known as *horizontal waste*, and only a more efficient exploiting of ILP can face this problem.

A strong debate is going on between ILP pessimists and optimists. ILP advocates hold that ILP is abudant, and can be exploited with a few more ten millions of transistors, and a little compiler magic (see for instance Itanium and Sparc 64 V projects). Yale Patt at University of Texas and John Shen at Carnegie-Mellon University believe that advanced superscalar techniques, such as static scheduling, prediction, trace-processing and superspeculation, will allow to ILP to scale suitably: in their opinion a 16- or 32-wide superscalar processor will sustain a ILP of more than 10 instructions per cycle.

These processors will feature a very large monolithic core, along with the related complexity of design and testing. Explicit thread-level parallelism (TLP) can be seen as a way to keep processors simpler. Chip multiprocessors (CMP), as IBM Power4 and Sun MAJC trust in the parallelism that can be found between different threads. These projects will implement, on a single chip, a few replicated superscalar processors: there will be the opportunity to run multiple threads, each one on an independent complete advanced processor (with smaller execution bandwidth).

Nonetheless, both these kinds of architecture (ILP- or TLP-oriented) suffer from poor utilization if the work-load does not match the design parameters: they shows no flexibility when the parallelism moves from ILP to TLP or *vice versa*.

## SMT features

SMT combines superscalar execution with latency-hiding ability of multithreaded architectures . Design goals are:

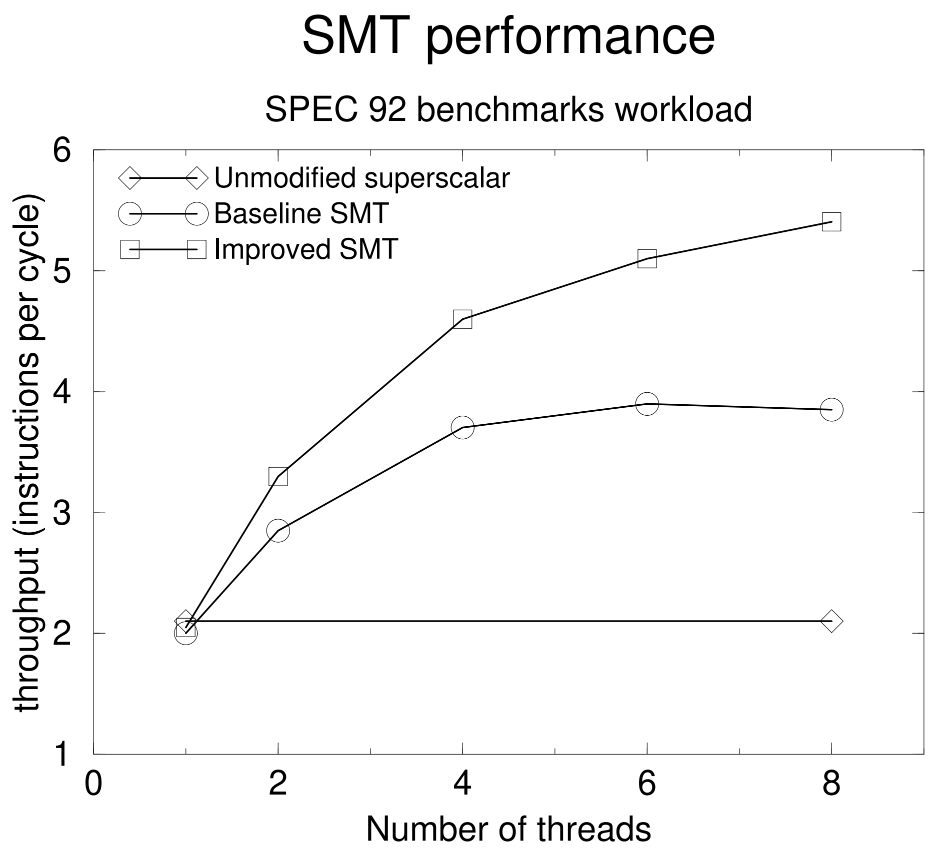
- minimal changes to existing superscalar processors,
- minimal effect on single-thread performance (2% ),
- high throughtput with many threads.

Features:

- no cost for context switching,
- synchronization on-chip,
- threads compete every cycle for resources (cache, functional units, renaming registers),
- solution to long latencies and limited ILP.

icount: choosing the best instruction from all threads.

# SMT performance



Performance of SMT (basic and advanced version)

## Livermore loops

Developed to *measure numerical computation rates for a spectrum of cpu-limited computational structures* (F. H. McMahon, 1972). Cpu-intensive, really different behaviors. Four main groups:

1. independent-iteration loops,
2. accumulation loops,
3. loop-carried-dependence loops,
4. larger (more complex) loops.

## Independent-iteration loops (1)

Interleaving useful in most cases. Average 65% processor utilization (up to 94%).

(a)

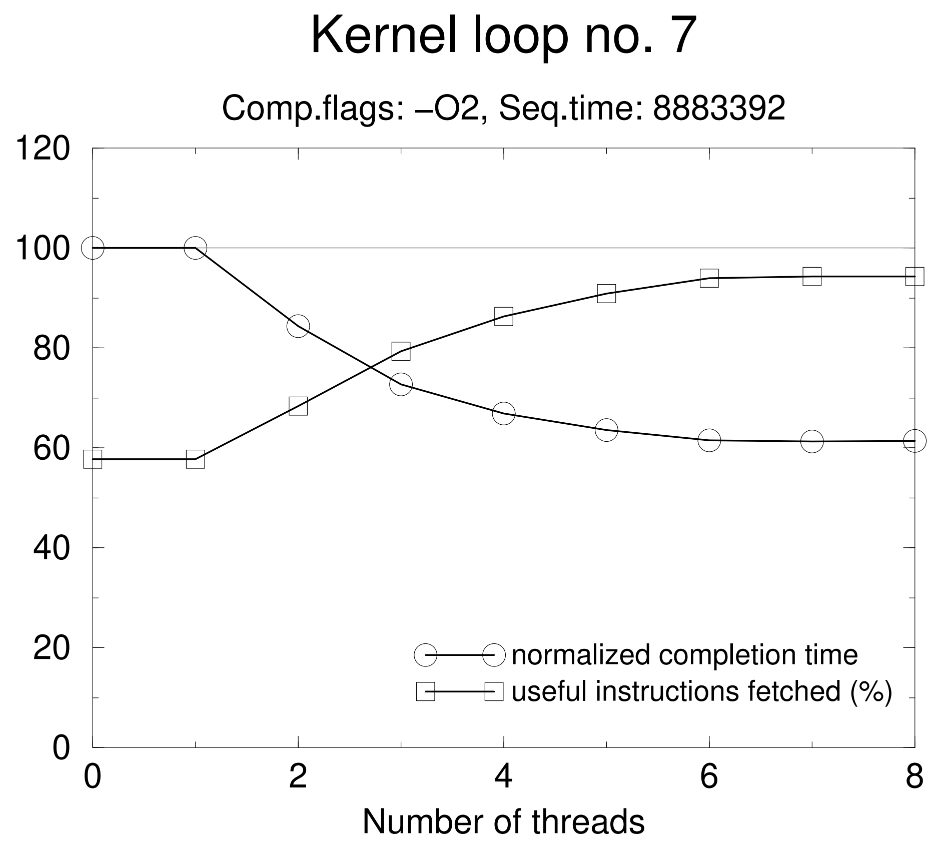
```
for ( k=0 ; k<n ; k++)
{
    a[k] = ....
}
```

(b)

```
for ( k=threadid ; k<n ; k += NUMTHREADS )
{
    a[k] = ...
}
```

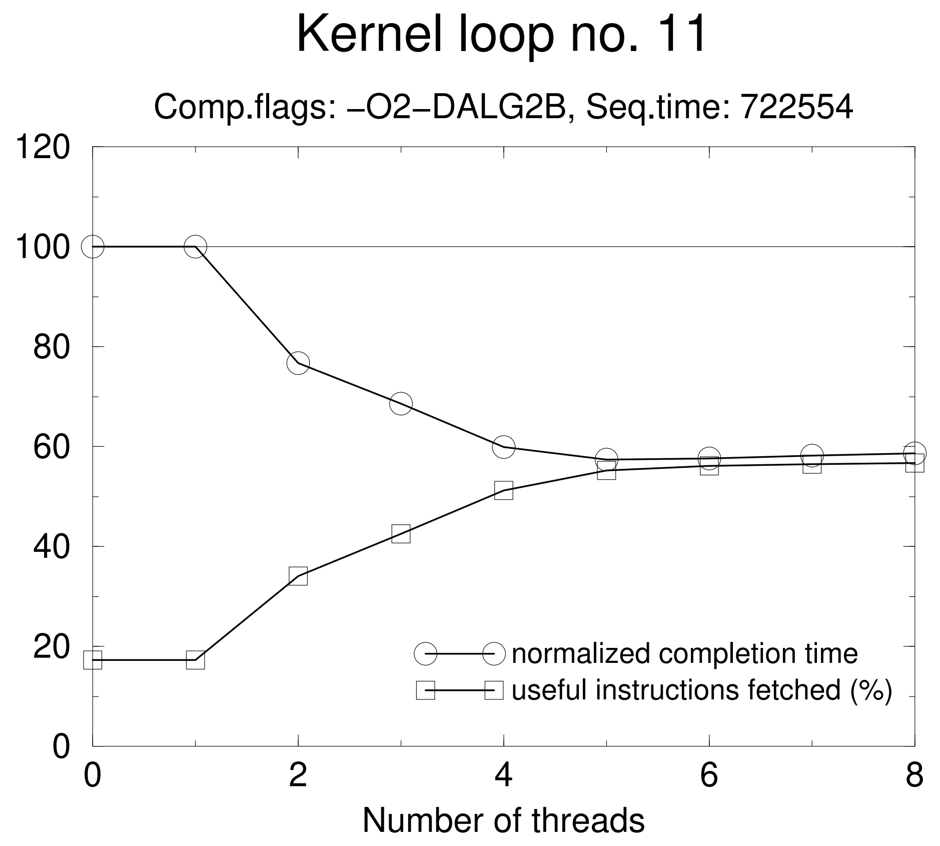
Original code (a) and MT version with interleaved iterations (b)

## Independent-iteration loops (2)




Processor utilization and normalized completion time.

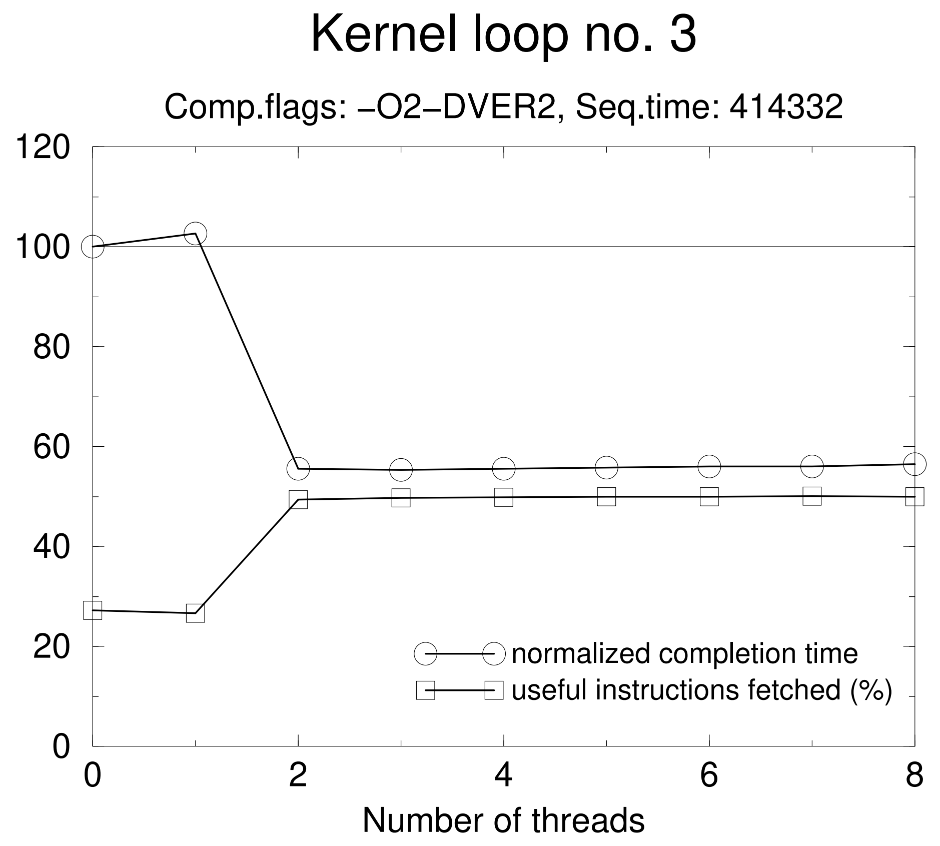
# Loop-carried-dependence loops




Cyclic reduction used successfully, loop skewing has a high overhead.

Advanced rewriting is needed. cyclic reduction algorithm

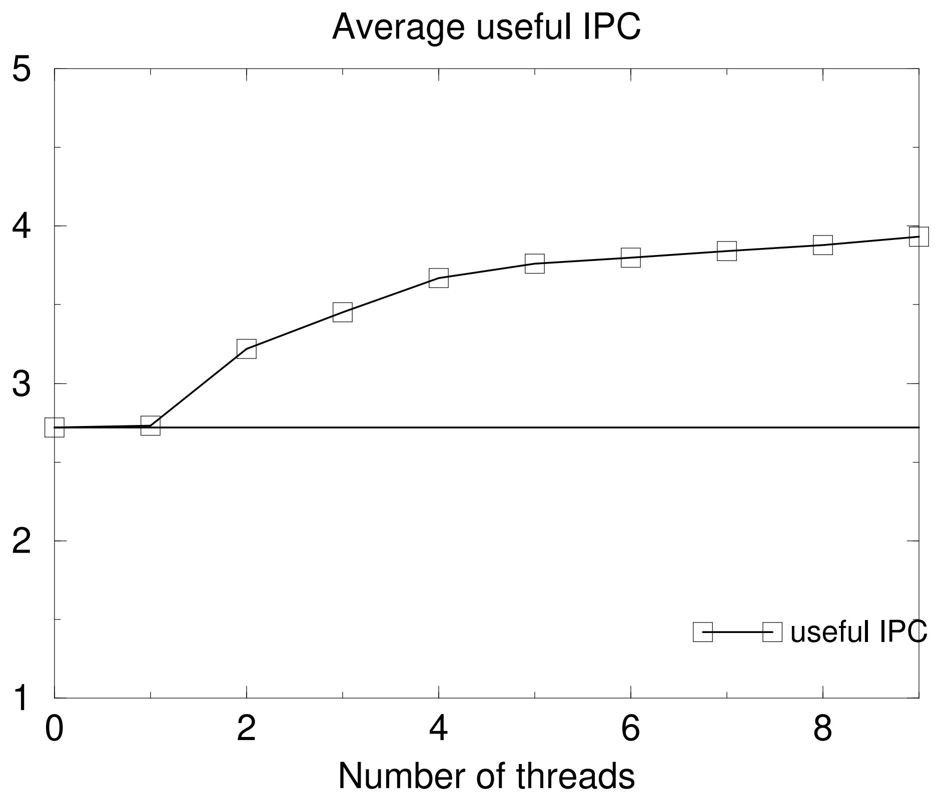
# Accumulation loops




Local accumulation allows good parallelization.

# Overall performance improvement

Performance gain




Effect on average instructions-per-cycle (IPC) rate

## Modeling performance with compile-time info.

**Goal:** to predict multithreading performance on SMT;

**Methodology:** for each block, the following data are collected:

- size,
- critical path length (CPL), loop-critical path (loopCPL),
- type and number of instructions.

A simple model uses them to predict improvement coming with multithreading and the best number of threads. It is based on architectural information: issue bandwidth, instruction-queue size, maximum number of hardware contexts.

# Computing CPL with Atom

| Assembler code | depends from | dep. delay | loop dep. delay |
|---|---|---|---|
| 0 (0x20005960) : ldt \$f1, 0(t1) | - | 0 | 0 |
| 1 (0x20005964) : ldt \$f10, 0(t2) | - | 0 | 0 |
| 2 (0x20005968) : subt \$f1,\$f10,\$f1 | 0,1 | 2 | 0 |
| 3 (0x2000596c) : addq t2, 0x8, t2 | 3 | 0 | 1 |
| 4 (0x20005970) : addq t4, 0x1, t4 | 4 | 0 | 1 |
| 5 (0x20005974) : addq t1, 0x8, t1 | 5 | 0 | 1 |
| 6 (0x20005978) : cmplt t4, a1, t0 | 4 | 1 | 1 |
| 7 (0x2000597c) : stt \$f1, 0(t3) | 2 | 6 | 0 |
| 8 (0x20005980) : addq t3, 0x8, t3 | 8 | 0 | 1 |
| 9 (0x20005984) : bne t0, 0x20005960 | 6 | 1 | 2 |
| | | CPL: | 6 |
| | | loopCPL: | 2 |

## Analytical model

- A new iteration cannot start before loopCPL cycles: at least loopCPL cycles per iteration (CPI) are needed. Saturation factor describes *functional-units overload*, as ratio of instructions to be executed vs. free slots.

$$best\ CPI_{loop} = loopCPL * saturation\ factor$$

- If more copies of loop-body can be queued together, latencies in the critical path can be hidden. More copies can be executed in the time required by one. This is another lower limit to cycles-per-iteration.

$$best\ CPI_{independent} = \frac{CPL * saturation\ factor}{copies}$$

## Results with gcc code

- With loops shorter than 40 instructions, 5% accuracy on predicted processor utilization.
- With loop-carried-dependence loops, average 1% accuracy (3% worst case).
- On average 6.4% accuracy.
- Bad results with larger loops, due to difficult-to-grasp behavior of the queueing mechanisms.

## Best number of threads

$$best\ \ copies = \begin{cases} \min_{1..5}\left(\frac{bandwidth_i}{number_i}\right) & \text{if indep. iterations} \\ 1 & \text{if dep. iterations} \end{cases}$$

| Kernel no. | 1 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Actual | 8 | 2 | 2 | 1 | 4 | 7 | 8 |
| Expected | 8.8 | 5.1 | 5.1 | 1 | 17.4 | 6.3 | 5.5 |

| Kernel no. | 9 | 10 | 11 | 12 | 13 | 21 | 23 |
|---|---|---|---|---|---|---|---|
| Actual | 8(16) | 7 | 1 | 2 | 5 | 3 | 1 |
| Expected | 35.3 | 9.2 | 1 | 4 | 3.3 | 6.0 | 1 |

## Conclusion

- Some templates come out to be general and useful (*iteration-interleaving, local accumulation, loop-fusion*).
- 1.3 more IPC, with general techniques; up to 3x speedup in some cases.
- The model is quite good with small loops (5% accuracy), and loop-carried-dependences loops (1% accuracy).
- Determination of the best number of threads gives an understanding of what is the trend.

## Future work

- Algorithmical implementation of the discussed parallel templates — an automatic rewriting tool before actual compiling (hints from MTA compiler).
- Improvement of the model (eg. phased behavior...), in order to cover more complex and larger loops.
- Improvement of the CPL algorithm, in order to make it work with a broader class of loops (eg. dataflow techniques as MURZ...).
- Registers are still needed? Broader architectural problem (doctorate?).

# References


(1) K. Diefendorff. Compaq chooses smt for alpha. *Microprocessor Report*, 13(16):1, 6–11, December 1999.

(2) J. L. Lo, S. J. Eggers, J. S. Emer, H. M. Levy, R. L. Stamm, and D. M. Tullsen. Converting thread-level parallelism into instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems*, pages 322–354, August 1997.

(3) J. L. Lo, S. J. Eggers, H. M. Levy, S. S. Parekh, and D. M. Tullsen. Tuning compiler optimizations for simultaneous multithreading. 30th Annual International Symposium on Microarchitecture (Micro-30), December 1997.

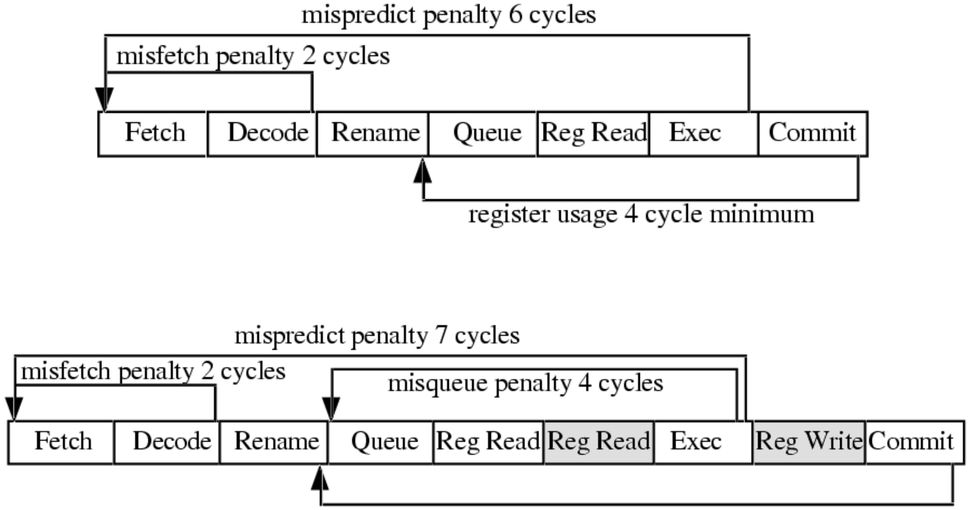
(4) D. M. Tullsen. Simulation and modeling of a simultaneous

multithreading processor. 22nd Annual Computer Measurement Group Conference, December 1996.

(5) D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. 23rd Annual International Symposium on Computer Architecture, Philadelphia, PA, May 1996. Reprinted in Readings in Computer Architecture.

(6) D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, Italy, June 1995. Reprinted in 25 Years of the International Symposia on Computer Architecture: Selected Papers, 1998.

# Pipeline overhead




Original MIPS R10000 pipeline

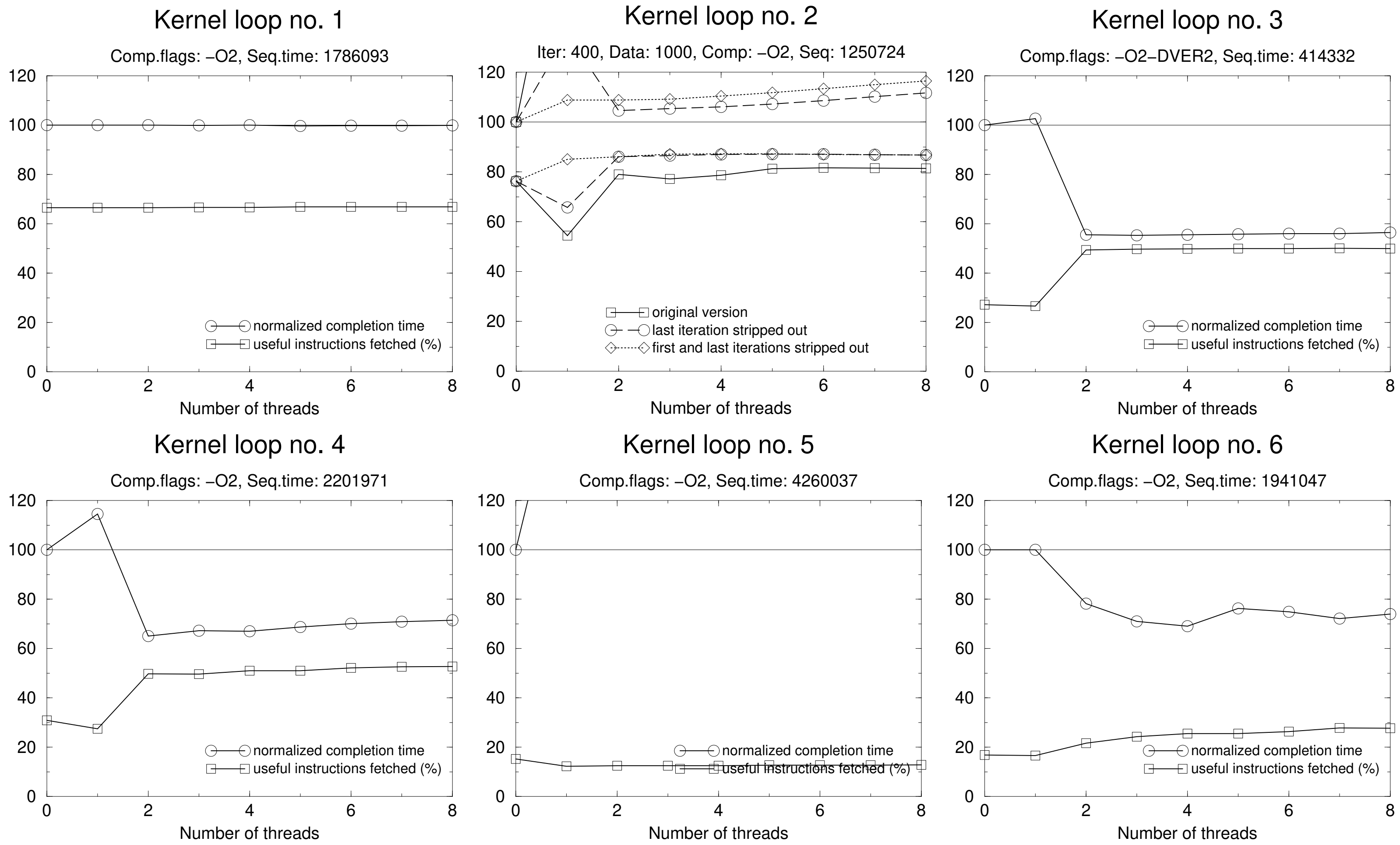
Basic SMT architecture pipeline

$\Rightarrow$ small overhead of two cycles for the larger register file, degradation in single-thread performance (2% ).

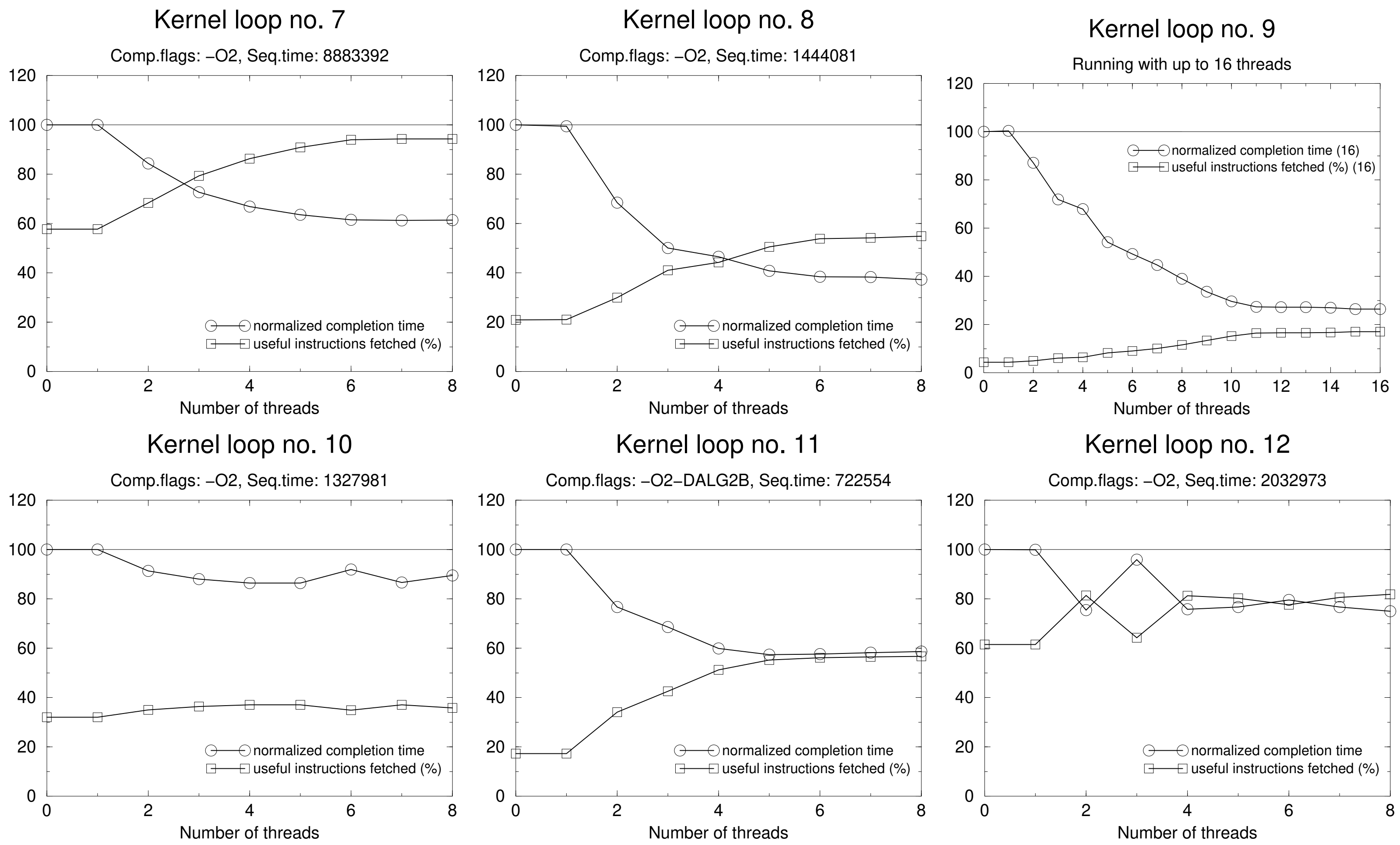
# Detailed structures' description

| functional block | R10000-2x | SMT |
|---|---|---|
| remapping tables (register renaming) | $O(d^2)$ | $O(t * \log_2(t))$ |
| enlarged INT and FP register files | $O(d^2)$ | $O(t)$ |
| fetch block, multiple PC and *icount* fetch policy | $O(d)$ | $O(t)$ |
| branch predict block, return stack | $O(d)$ | $O(t)$ |
| instruction squash ($O^3$) | N/A | $O(t)$ |
| instruction commit ($O^3$) | $O(d^2)$ | $O(t)$ |
| free list with wider tag storage ($O^3$) | $O(d^2)$ | $O(\log_2(t))$ |
| instruction queue ($O^3$) | $O(d^2)$ | $O(\log_2(t))$ |
| recover *icount* ($O^3$) | N/A | $O(\log_2(t))$ |
| dtag, itag | $O(\log_2(d))$ | $O(\log_2(t))$ |
| routing | $O(d)$ | $O(\log_2(t))$ |

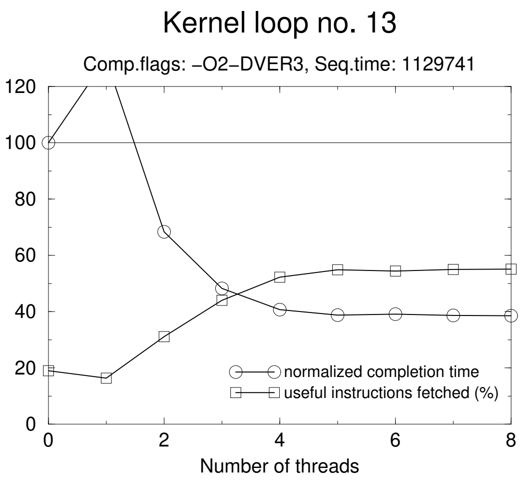
# All the plots

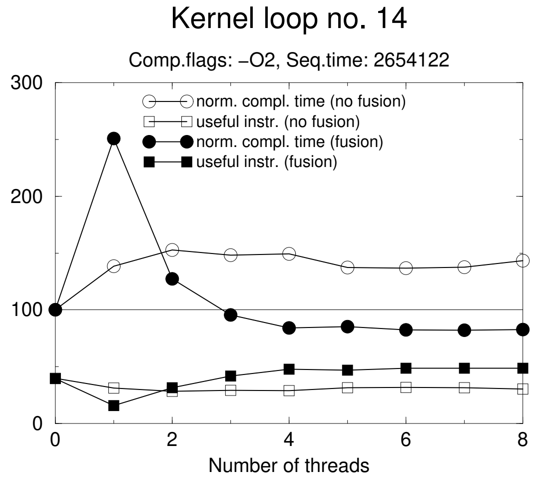


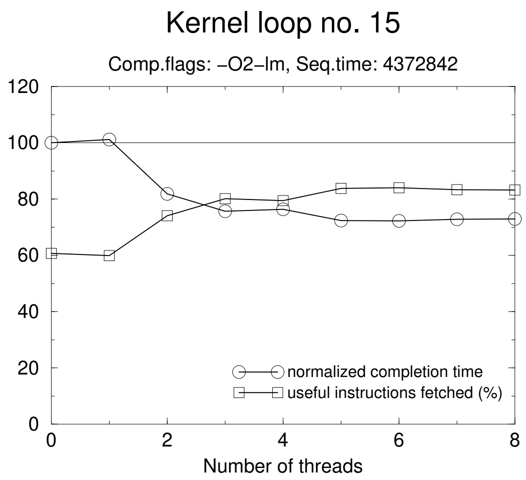
# All the plots (2)

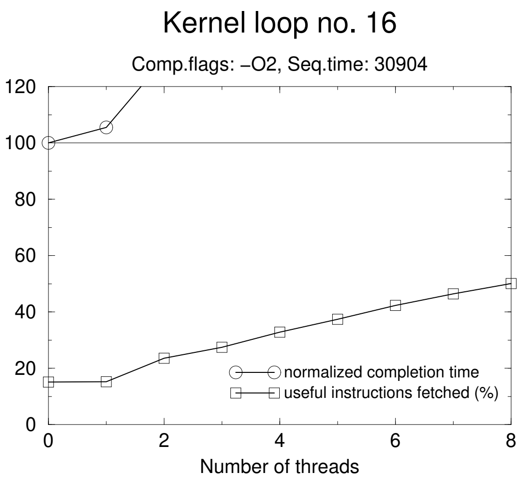


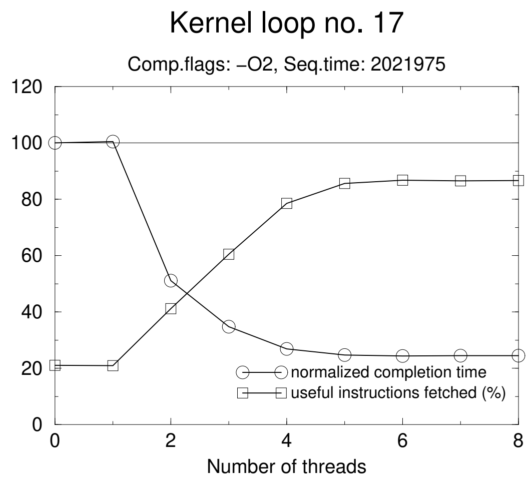
# All the plots (3)

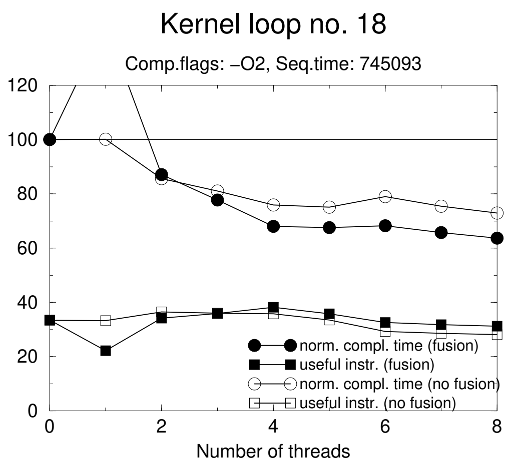


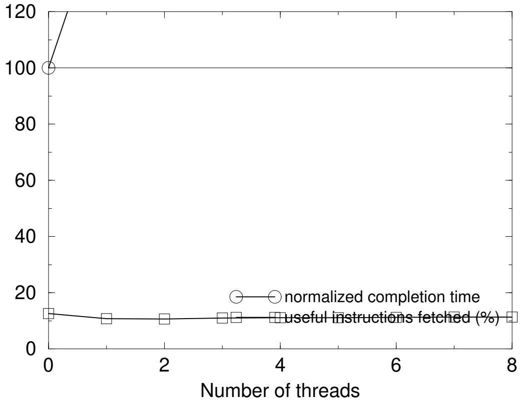



# All the plots (4)

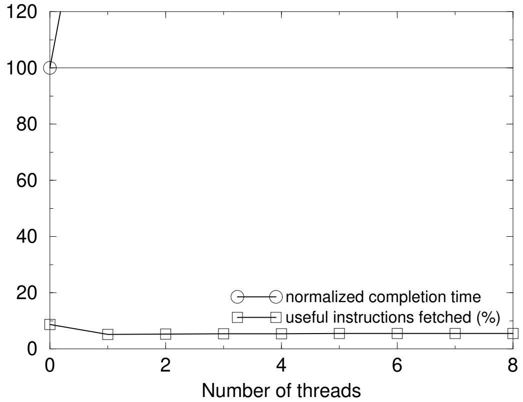
### Kernel loop no. 19

Comp.flags: –O2, Seq.time: 319934



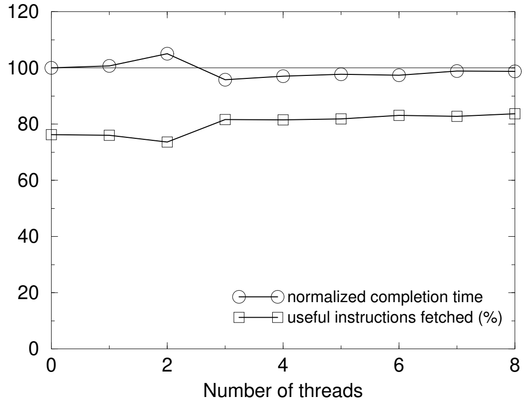

### Kernel loop no. 20

Comp.flags: –O2, Seq.time: 194559



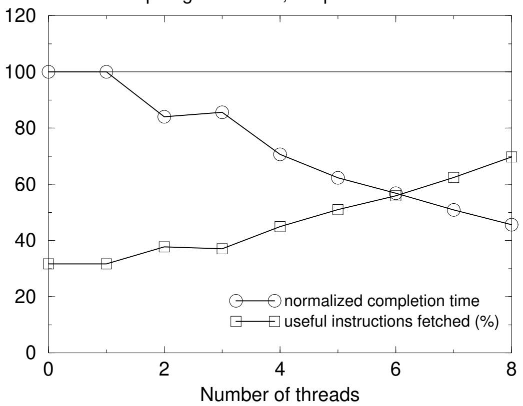

### Kernel loop no. 21

Comp.flags: –O2, Seq.time: 1342392



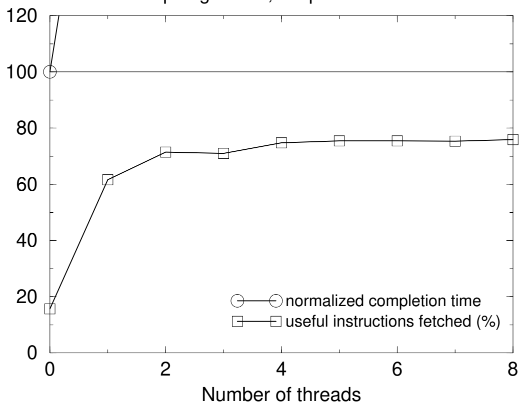

### Kernel loop no. 22

Comp.flags: –lm–O2, Seq.time: 1931250



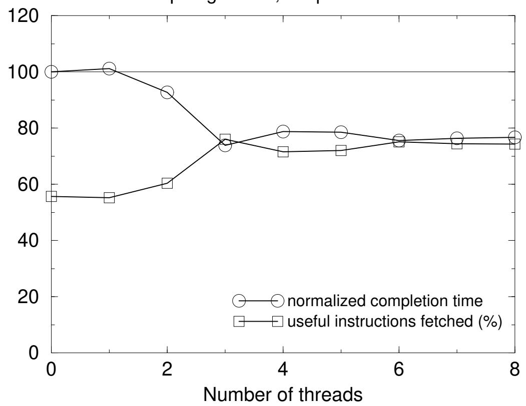

### Kernel loop no. 23

Comp.flags: –O2, Seq.time: 1313329




### Kernel loop no. 24

Comp.flags: –O2, Seq.time: 225033



## Analytical model

Performance limited by loopCPL:

$$saturation\ factor = \max\left(1, \max_{1..5}\left(\frac{number_i}{loopCPL * bandwidth_i}\right)\right)$$

$$best\ \ CPI_{loop} = loopCPL * saturation\ factor$$

$$best\ proc.\ util._{loop} = \frac{size}{loopCPL * saturation\ factor * BW}$$

Performance limited by CPL and number of copies:

$$saturation\ factor = \max\left(1, \max_{1..5}\left(\frac{number_i * copies}{CPL * bandwidth_i}\right)\right)$$

$$min.\ copies = \min\left(\frac{\#int.instr.}{32}, \frac{\#FPinstr.}{32}\right)$$

$$max.\ copies = \max\left(\frac{\#int.instr.}{32}, \frac{\#FPinstr.}{32}\right)$$

$$worst\ CPI_{independent} = \frac{CPL * saturation\ factor_{min}}{min.copies}$$

$$worst\ proc.\ util._{independent} = \frac{size * min.copies}{CPL * saturation\ factor_{min} * BW}$$

$$best\ CPI_{independent} = \frac{CPL * saturation\ factor_{max}}{max.copies}$$

$$best\ proc.\ util._{independent} = \frac{size * max.copies}{CPL * saturation\ factor_{max} * BW}$$

# Results of modeling (gcc code)

| kernel no. | type | size | CPL | loopCPL | actual performance | best proc. util. loop | worst proc. util. ind. | best proc. util. ind. | expected MIN | expected MAX |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | IND | 19 | 21 | 2 | 66.52 | 100.00 | 25.85 | 72.38 | 25.85 | 72.38 |
| 3 | IND | 9 | 6 | 4 | 27.06 | 28.12 | 85.71 | 96.43 | 28.12 | 28.12 |
| 4 | IND | 9 | 6 | 4 | 29.81 | 28.12 | 85.71 | 96.43 | 28.12 | 28.12 |
| 5 | LOOP | 13 | | 10 | 15.20 | 16.25 | | | 16.25 | 16.25 |
| 6 | IND | 13 | 32 | 8 | 16.82 | 20.31 | 14.77 | 81.25 | 14.77 | 20.31 |
| 7 | IND | 41 | 34 | 2 | 57.75 | 96.09 | 19.29 | 30.15 | 19.29 | 30.15 |
| 8 | IND | 122 | 84 | 2 | 21.09 | 100.00 | 7.00 | 14.90 | 7.00 | 14.90 |
| 9 | IND | 32 | 200 | 2 | 4.33 | 70.59 | 3.76 | 4.27 | 3.76 | 4.27 |
| 10 | IND | 34 | 46 | 2 | 32.69 | 85.00 | 11.83 | 32.85 | 11.83 | 32.85 |
| 11 | LOOP | 10 | | 7 | 17.41 | 17.86 | | | 17.86 | 17.86 |
| 12 | IND | 10 | 6 | 2 | 61.52 | 62.50 | 74.07 | 83.33 | 62.50 | 62.50 |
| 13 | IND | 86 | 38 | 6 | 19.48 | 93.48 | 13.12 | 53.25 | 13.12 | 53.25 |
| 21 | IND | 13 | 11 | 2 | 76.68 | 81.25 | 42.98 | 88.64 | 42.98 | 81.25 |
| 23 | LOOP | 33 | | 34 | 15.71 | 12.13 | | | 12.13 | 12.13 |