

*Optimization Techniques for
Complex Memory Hierarchies –
the Evolution of Superscalar
GEMM-based level 3 BLAS*

Fred Gustavson^{*}, Isak Jonsson[†], Bo Kågström[†], Per Ling[†]

^{*}gustav@watson.ibm.com, IBM T.J. Watson Research Center, P.O. Box 218,
Yorktown Heights, NY 10598, U.S.A.

[†]{isak,bokg,pol}@cs.umu.se, Department of Computing Science and
HPC2N, Umeå University, S-901 87 Umeå, Sweden

Outline

- Superscalar GEMM-based level 3 BLAS
(Kågström, Ling)
- Enhanced techniques
 - Blocking and recursive algorithms
 - Kernels
 - Prefetching
 - Data structures
 - Dynamic load balancing
- Some performance results

Superscalar GEMM-based level 3 BLAS

Objectives: Portability and High performance

Concept: Level 3 BLAS reorganized to be rich in _GEMM operations ($C \leftarrow \alpha AB + \beta C$)

- Emerged from GEMM-based level 3 BLAS (Kågström, Ling, Van Loan)
- Only one highly optimized level 3 BLAS routine required, thus limited effort for improved and new architectures.
- “In-line” unrolled code for level 1 and 2 operations.

Impact: Used by NEC, ASCI-red, ATLAS, PHiPAC and more.

Recursive GEMM

If $M \leq \Gamma$ and $N \leq \Gamma$ and $K \leq \Gamma$

 solve problem using optimized kernel subroutine.

Otherwise,

 If $M = \max(M, N, K)$,

 two recursive calls: $\underline{M'} = \underline{M/2}$, $N' = N$, $K' = K$.

 else, if $N = \max(N, K)$,

 two recursive calls: $M' = M$, $\underline{N'} = \underline{N/2}$, $K' = K$.

 else,

 two recursive calls: $M' = M$, $N' = N$, $\underline{K'} = \underline{K/2}$.

Why $\Gamma > 1$? For very small problems, the overhead for the recursion becomes too expensive – use *optimized kernel*.

Blocking

Recursive blocking \Rightarrow automatic and “squarish”.

Automatic: Cache fitting. If a problem does not fit in cache, it is solved as two similar smaller problems etc.

Squarish: The property $MNK \gg MN + NK + MK$ is maintained, which is a must for good data reuse, i.e., each element is an operand in many operations.

Automatic + Squarish \Rightarrow No explicit tuning w.r.t. memory hierarchy necessary except for kernel routine.

The GEMM kernel

- The innermost part – performs all computing! (Assumes data in cache.)
- Works on fixed dimensions \Rightarrow simple address arithmetics.
- Register blocking enables reuse of loaded information and makes overlapping of memory load instructions and floating point arithmetic instructions.
- We use a fine tuned 4×4 register blocking.

Prefetching in kernels

Prefetching – a technique to overcome load latency.

Prefetching means that “hint” instructions are inserted to initiate transactions in the memory hierarchy, e.g., between level 2 cache and level 1 cache. Such instructions are not sufficiently supported in programming languages.

Imposes assembly coding. Approaches:

1. The high-level source code is compiled to assembly code, which is then modified.
2. The code is written in assembly code from the beginning.

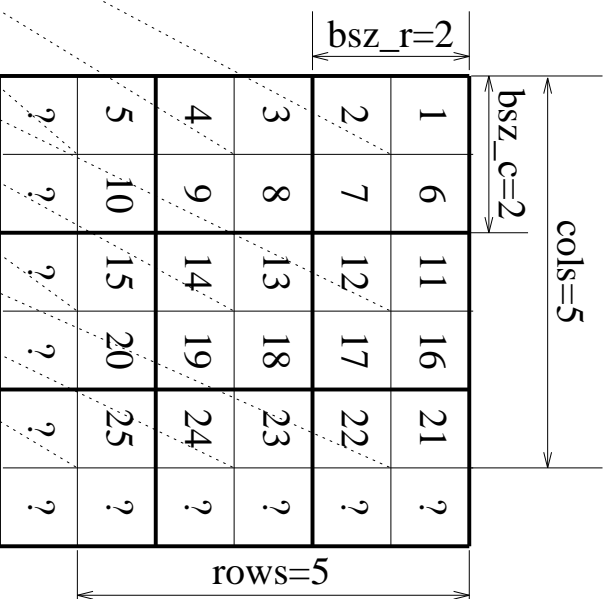
Algorithmic prefetching

Make use of the fact that $C = C + A \times B$ takes $2n^3$ operations, but only uses $3n^2$ elements. Implicit loads of the next three sub-blocks are embedded into the code which multiplies the current three sub-blocks.

Example: For kernels which uses 16×16 -blocks, we hide the cost of loading $3 \cdot 16^2 = 768$ elements within the cost of executing $16^3 = 4096$ FMAs.

Data structures

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25



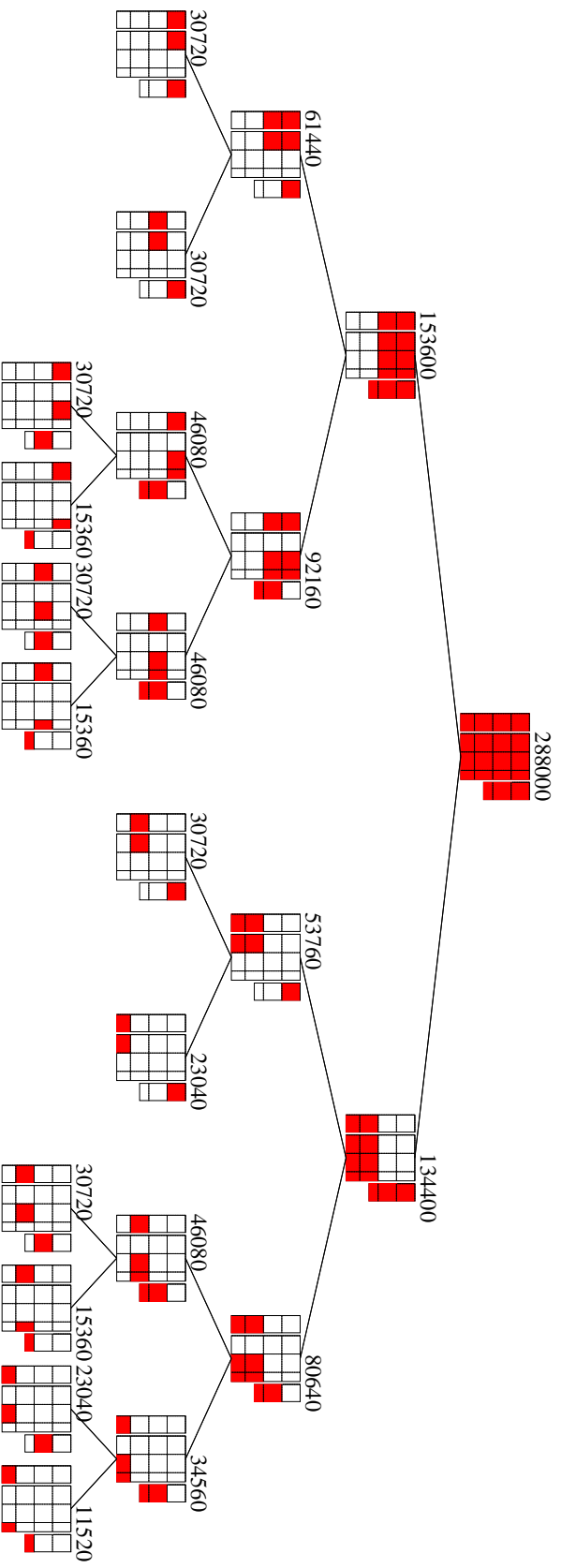
blockaddress:



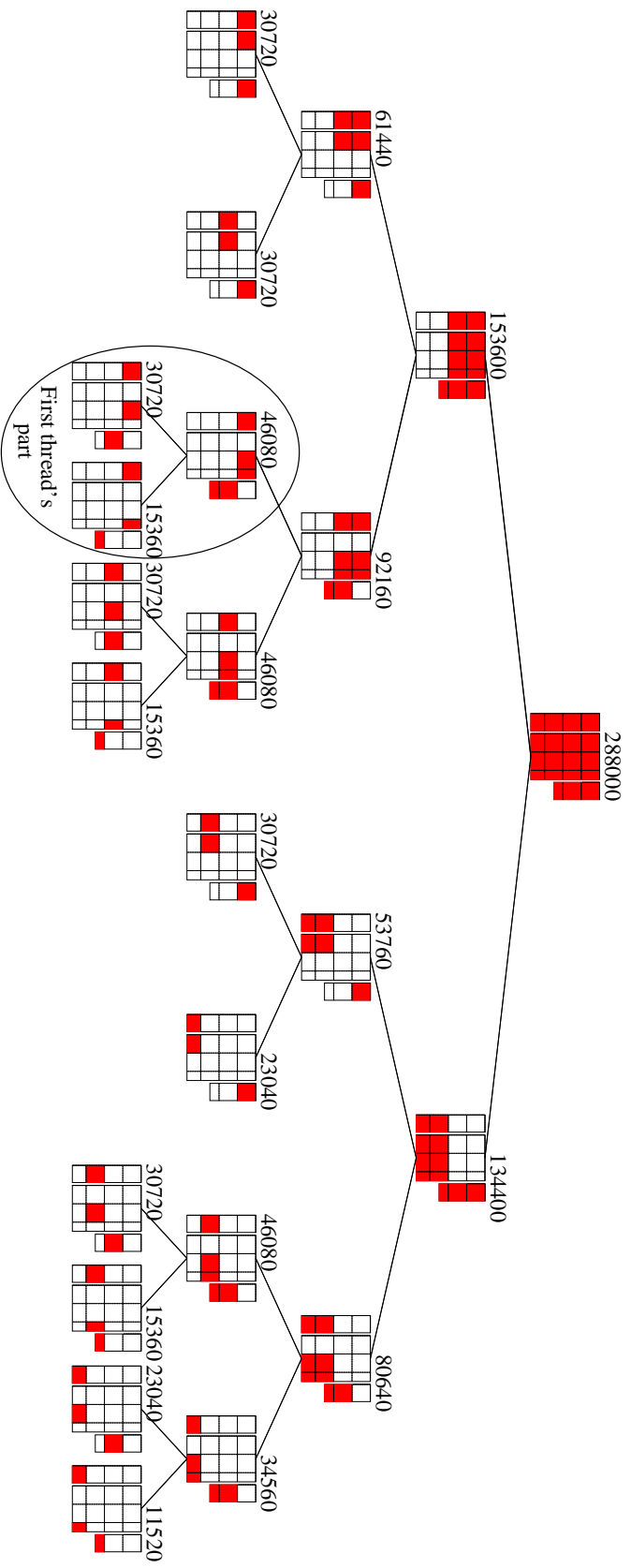
25	?	?	?	21	22	?	?	?	5	?	10	?	3	4	8	9	1	2	6	7	11	12	16	17	15	?	20	?	23	24	?	?	13	14	18	19
----	---	---	---	----	----	---	---	---	---	---	----	---	---	---	---	---	---	---	---	---	----	----	----	----	----	---	----	---	----	----	---	---	----	----	----	----

An arbitrary blocked format.

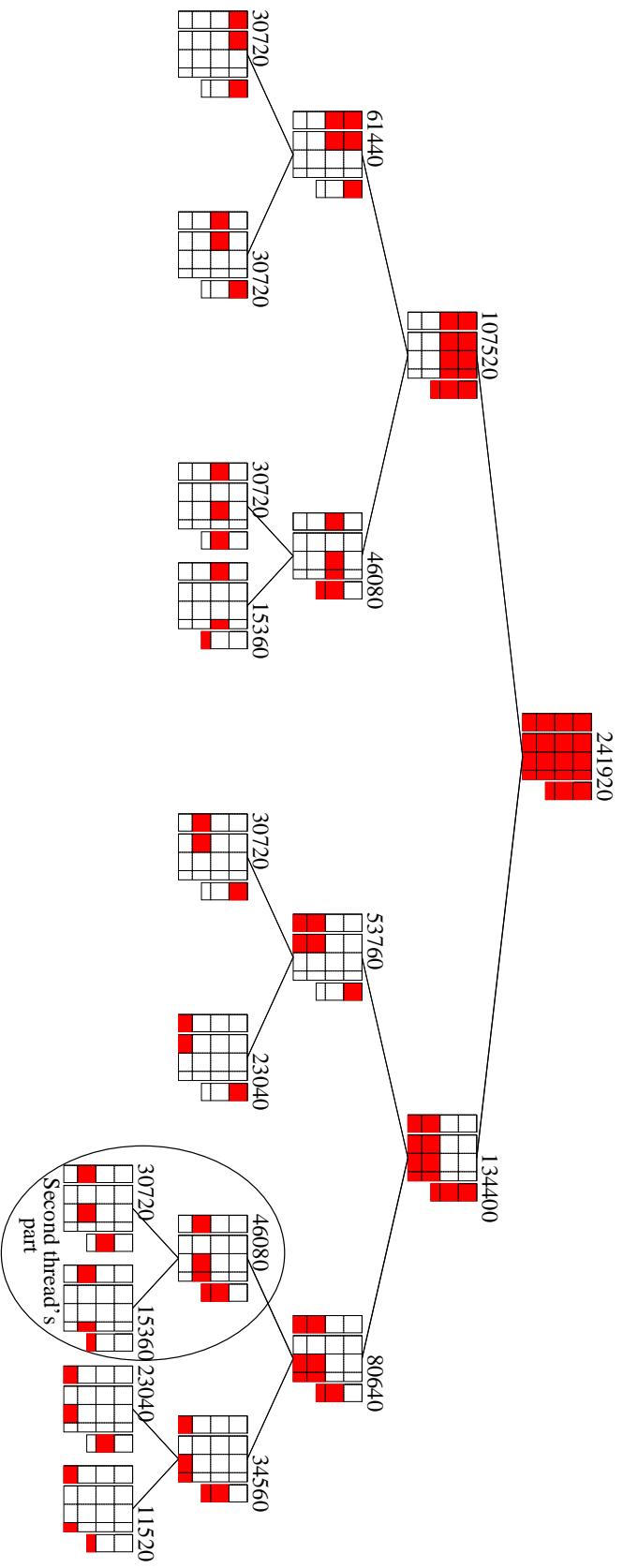
Dynamic load balancing (1/3)



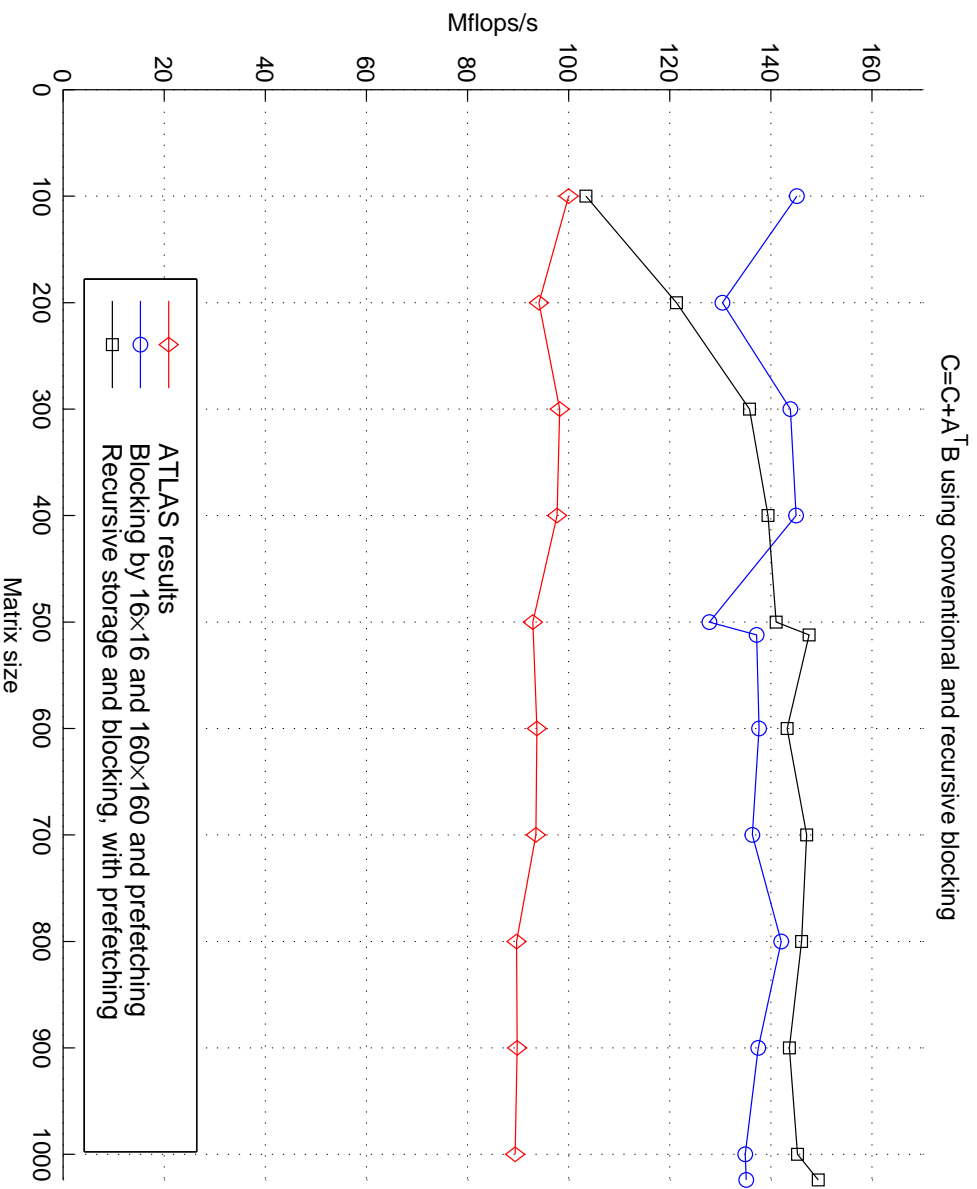
Dynamic load balancing (2/3)



Dynamic load balancing (3/3)



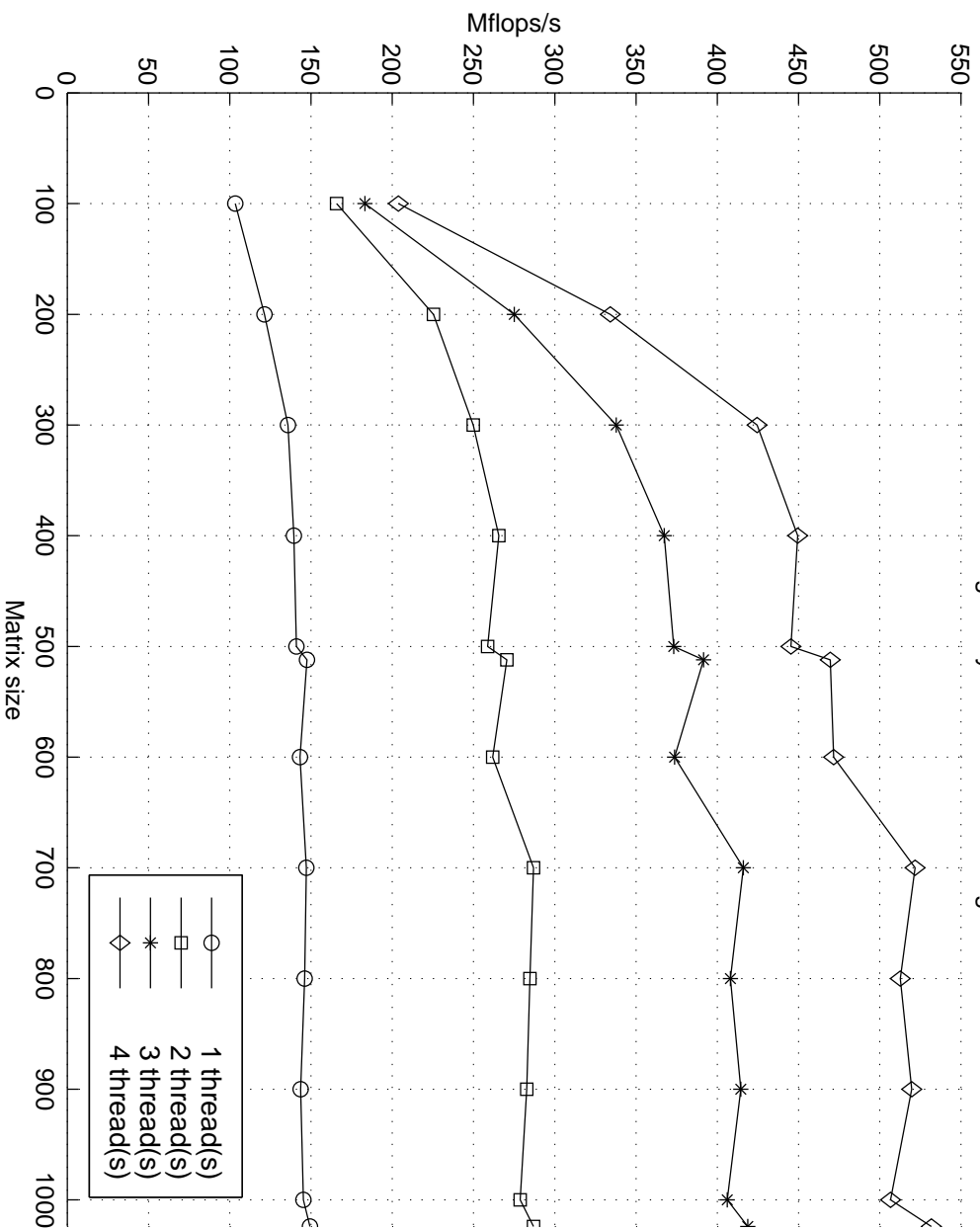
Uniprocessor results



(IBM SP High Node, 112 MHz 604, peak 224 Mflops/s)

Multiprocessor results

C=C+A^TB using the dynamic load balancing



Preliminary TRSM results

