A Comparison of Cache-conscious and Cache-oblivious Programs

Keshav Pingali, University of Texas, Austin

Joint work with
Kamen Yotov, Goldman Sachs
Tom Roeder, Cornell University
John Gunnels, IBM T.J. Watson Research Center
Fred Gustavson, IBM T.J.Watson Research Center

Memory Hierarchy Management

- Cache-conscious (CC) approach:
  - Blocked iterative algorithms and arrays (usually)
    - Code and data structures have parameters that depend on careful blocking for memory hierarchy
  - Used in dense linear algebra libraries: BLAS, LAPACK
    - Lots of algorithmic data reuse: $O(N^3)$ operations on $O(N^2)$ data
- Cache-oblivious (CO) approach:
  - Recursive algorithms and data structures (usually)
    - Not aware of memory hierarchy: approximate blocking
    - I/O optimal: Hong and Kung, Frigo and Leiserson
  - Used in FFT implementations: FFTW
    - Little algorithmic data reuse: O(N(logN)) computations on O(N) data

Questions

- Does CO approach perform as well as CC approach?
  - Intuitively, a "self-adaptive" program that is oblivious of some hardware feature should be at a disadvantage.
  - Little experimental data in the literature
    - CO community believes their approach outperforms CC approach
    - But most studies from CO community compare performance with unblocked (unoptimized) CC codes
- If not, what can be done to improve the performance of CO programs?

One study

Piyush Kumar (LNCS 2625)

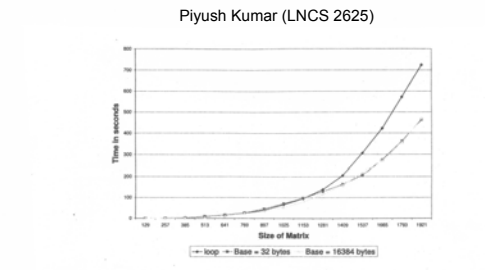


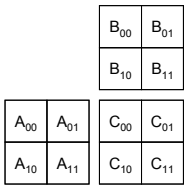
Fig. 9.3. Blocked cache oblivious matrix multiplication experiment.

- Studied recursive and iterative MMM on Itanium-2
- Recursive performs better
- But look at MFlops: 30 MFlops
- Intel MKL: 6GFlops

Organization of talk

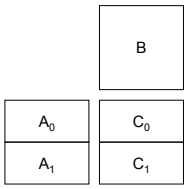
- CO and CC approaches to blocking
  - control structures
  - data structures
- Non-standard view of blocking (or why CO may work well)
  - reduce bandwidth required from memory
- Experimental results
  - UltraSPARC IIIi
  - Itanium
  - Xeon
  - Power 5
- Lessons and ongoing work

Cache-Oblivious Algorithms



$C_{00} = A_{00}*B_{00} + A_{01}*B_{10}$
$C_{01} = A_{01}*B_{11} + A_{00}*B_{01}$
$C_{11} = A_{11}*B_{01} + A_{10}*B_{01}$
$C_{10} = A_{10}*B_{00} + A_{11}*B_{10}$

- Divide all dimensions (AD)
- 8-way recursive tree down to 1x1 blocks
- Bilardi, et. al.
  - Gray-code order promotes reuse
- We use AD in rest of talk

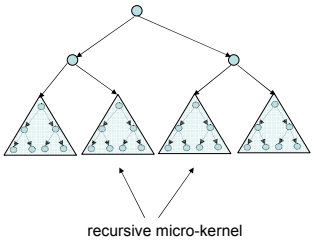


$C_0 = A_0*B$
$C_1 = A_1*B$

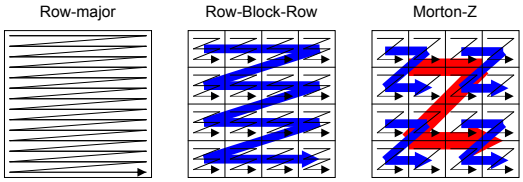
- Divide largest dimension (LD)
- Two-way recursive tree down to 1x1 blocks
- Frigo, Leiserson, et. al.

CO: recursive micro-kernel

- Internal nodes of recursion tree are recursive overhead; roughly
  - 100 cycles on Itanium-2
  - 360 cycles on UltraSPARC IIIi
- Large overhead: for LD, roughly one internal node per leaf node
- Solution:
  - Micro-kernel: code obtained by complete unrolling of recursive tree for some fixed size problem (RUxRUxRU)
  - Cut off recursion when sub-problem size becomes equal to micro-kernel size, and invoke micro-kernel
  - Overhead of internal node is amortized over micro-kernel, rather than a single multiply-add
  - Choice of RU: empirical

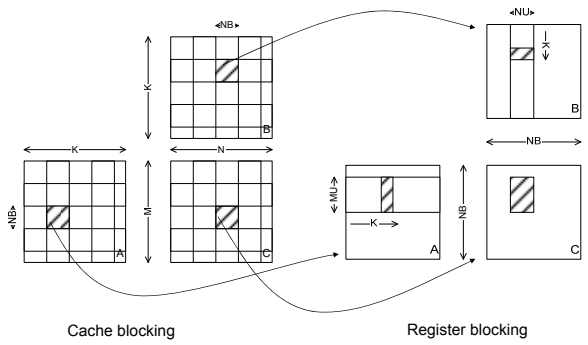


Data Structures



- Match data structure layout to access patterns
- Improve
  - Spatial locality
  - Streaming
- Morton-Z is more complicated to implement
  - Payoff is small or even negative in our experience
- Rest of talk: use RBR format with block size matched to microkernel

Cache-conscious algorithms



CC algorithms: discussion

- Iterative codes
  - Nested loops
- Implementation of blocking
  - Cache blocking
    - Mini-kernel: in ATLAS, multiply NBxNB blocks
    - Choose NB so $NB^2 + NB + 1 <= C_{L1}$
  - Register blocking
    - Micro-kernel: in ATLAS, multiply MUx1 block of A with 1xNU block of B into MUxNU block of C
    - Choose MU,NU so that MU + NU +MU*NU <= NR

Organization of talk

- CO and CC approaches to blocking
  - control structures
  - data structures
- Non-standard view of blocking
  - reduce bandwidth required from memory
- Experimental results
  - UltraSPARC IIIi
  - Itanium
  - Xeon
  - Power 5
- Lessons and ongoing work

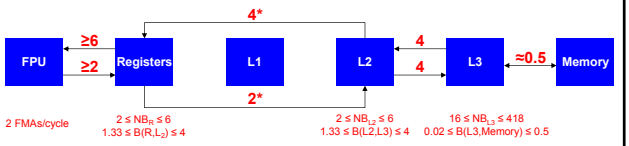
Blocking

- Microscopic view
  - Blocking reduces expected latency of memory access
- Macroscopic view
  - Memory hierarchy can be ignored if
    - memory has enough bandwidth to feed processor
    - data can be pre-fetched to hide memory latency
  - Blocking reduces bandwidth needed from memory
- Useful to consider macroscopic view in more detail

Blocking for MMM

- Assume processor can perform 1 FMA every cycle
- Ideal execution time for NxN MMM = N^3 cycles
- Square blocks: NB x NB
- Upper bound for NB:
  - working set for block computation must fit in cache
  - size of working set depends on schedule: at most $3NB^2$
  - Upper bound on NB: $3NB^2 \leq$ Cache Capacity
- Lower bound for NB:
  - data movement in block computation = $4\ NB^2$
  - total data movement $\leq (N / NB)^3 * 4\ NB^2 = 4\ N^3 / NB$ doubles
  - required bandwidth from memory = $(4\ N^3 / NB) / (N^3) = 4 / NB$ doubles/cycle
  - Lower bound on NB: 4/NB $\leq$ Bandwidth between cache and memory
- Multi-level memory hierarchy: same idea
  - sqrt(capacity(L)/3) $\geq NB_L \geq$ 4/Bandwidth(L,L+1) (levels L,L+1)

Example: MMM on Itanium 2



* Bandwidth in doubles per cycle; Limit 4 accesses per cycle between registers and L2

- Between L3 and Memory
  - Constraints
    - $8 / NB_{L3} \leq 0.5$
    - $3 * NB_{L3}^2 \leq 524288$ (4MB)
  - Therefore Memory has enough bandwidth for $16 \leq NB_{L3} \leq 418$
    - $NB_{L3} = 16$ required $8 / NB_{L3} = 0.5$ doubles per cycle from Memory
    - $NB_{L3} = 418$ required $8 / NB_R \approx 0.02$ doubles per cycle from Memory
    - $NB_{L3} > 418$ possible with better scheduling

Lessons

- Reducing bandwidth requirements
  - Block size does not have to be exact
  - Enough for block size to lie within an interval that depends on hardware parameters
  - If upper bound on NB is more than twice lower bound, divide and conquer will automatically generate a block size in this range

  ➔ approximate blocking CO-style is OK
- Reducing latency
  - Accurate block sizes are better
  - If block size is chosen approximately, may need to compensate with prefetching

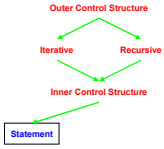
Organization of talk

- Non-standard view of blocking
  - reduce bandwidth required from memory
- CO and CC approaches to blocking
  - control structures
  - data structures
- Experimental results
  - UltraSPARC IIIi
  - Itanium
  - Xeon
  - Power 5
- Lessons and ongoing work

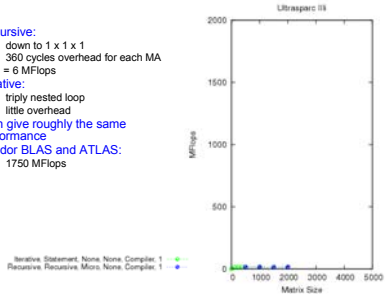
UltraSPARC IIIi

- Peak performance: 2 GFlops (1 GHZ, 2 FPUs)
- Memory hierarchy:
  - Registers: 32
  - L1 data cache: 64KB, 4-way
  - L2 data cache: 1MB, 4-way
- Compilers
  - C: SUN C 5.5

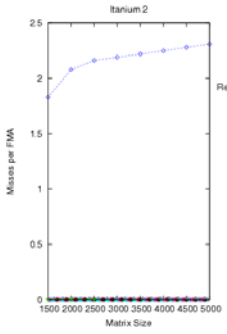
Naïve algorithms



- Recursive:
  - down to 1 x 1 x 1
  - 360 cycles overhead for each MA = 6 MFlops
- Iterative:
  - triply nested loop
  - little overhead
- Both give roughly the same performance
- Vendor BLAS and ATLAS:
  - 1750 MFlops

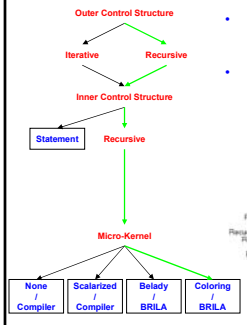


Miss ratios

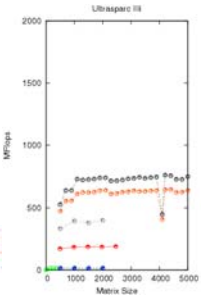


- Misses/FMA for iterative code is roughly 2
- Misses/FMA for recursive code is 0.002
- Practical manifestation of theoretical I/O optimality results for recursive code
- However, two competing factors affect performance:
  - cache misses
  - overhead
- 6 MFlops is a long way from 1750 MFlops!

Recursive micro-kernel



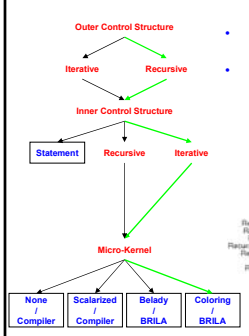
- Recursion down to RU(=8)
  - Unfold completely below RU to get a basic block
- Micro-Kernel
  - Scheduling and register allocation using heuristics for large basic blocks in BRILA compiler



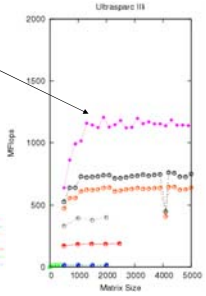
Lessons

- Bottom-line on UltraSPARC:
  - Peak: 2 GFlops
  - ATLAS: 1.75 GFlops
  - Best CO strategy: 700 MFlops
- Similar results on other machines:
  - Best CO performance on Itanium: roughly 2/3 of peak
- Conclusion:
  - Recursive micro-kernels are not a good idea

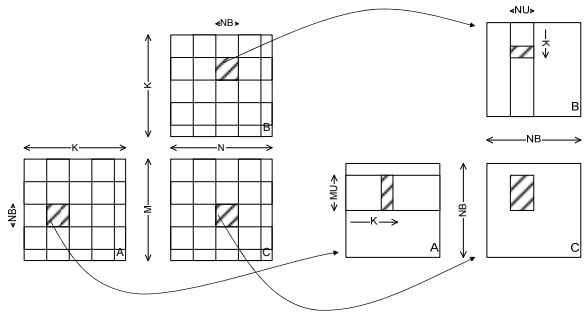
Recursion + Iterative micro-kernel



- Recursion down to MU x NU x KU (4x4x120)
- Micro-Kernel
  - Completely unroll MU x NU nested loop
  - Construct a preliminary schedule
  - Perform Graph Coloring register allocation
  - Schedule using BRILA compiler



Iterative micro-kernel



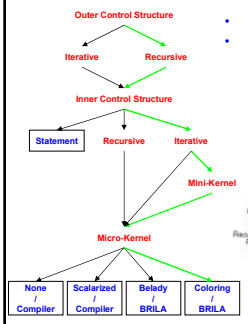
Cache blocking

Register blocking

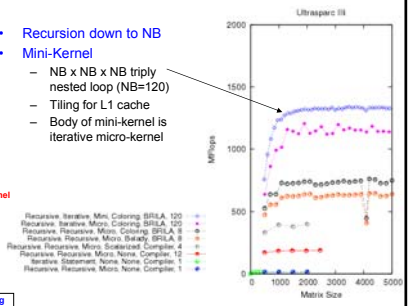
Lessons

- Two hardware constraints on size of micro-kernels:
  - I-cache limits amount of unrolling
  - Number of registers
- Iterative micro-kernel: three degrees of freedom (MU,NU,KU)
  - Choose MU and NU to optimize register usage
  - Choose KU unrolling to fit into I-cache
- Recursive micro-kernel: one degree of freedom (RU)
  - But even if you choose rectangular tiles, all three degrees of freedom are tied to both hardware constraints

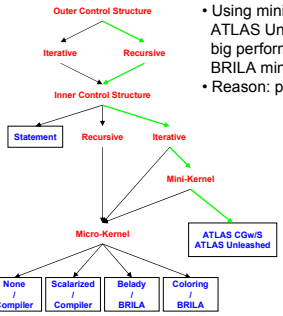
Recursion + mini-kernel



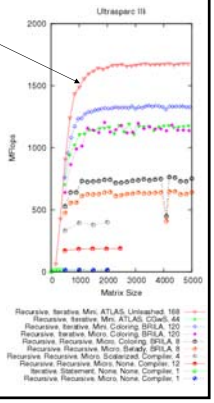
- Recursion down to NB
- Mini-Kernel
  - NB x NB x NB triply nested loop (NB=120)
  - Tiling for L1 cache
  - Body of mini-kernel is iterative micro-kernel



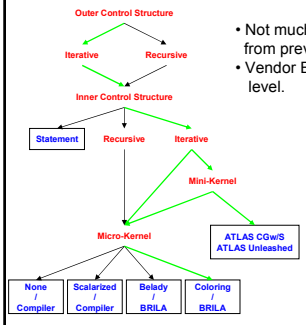
Recursion + mini-kernel + pre-fetching



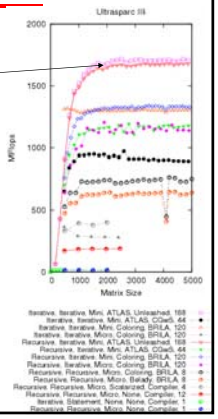
- Using mini-kernel from ATLAS Unleashed gives big performance boost over BRILA mini-kernel.
- Reason: pre-fetching



Vendor BLAS



- Not much difference from previous case.
- Vendor BLAS is at same level.



Lessons

- Vendor BLAS gets highest performance
- Pre-fetching boosts performance by roughly 40%
- Iterative code: pre-fetching is well-understood
- Recursive code: not well-understood

Summary

- Iterative approach has been proven to work well in practice
  - Vendor BLAS, ATLAS, etc.
  - But requires a lot of work to produce code and tune parameters
- Implementing a high-performance CO code is not easy
  - Careful attention to micro-kernel and mini-kernel is needed
- Using fully recursive approach with highly optimized recursive micro-kernel, we never got more than 2/3 of peak.
- Issues with CO approach
  - Recursive Micro-Kernels yield less performance than iterative ones using same scheduling techniques
  - Pre-fetching is needed to compete with best code: not well-understood in the context of CO codes

Ongoing Work

- Explain performance of all results shown
- Complete ongoing Matrix Transpose study
- Proteus system and BRILA compiler
- I/O optimality:
  - Interesting theoretical results for simple model of computation
  - What additional aspects of hardware/program need to be modeled for it to be useful in practice?

Miss ratios