Introduction

- Stream processor: high computation to bandwidth ratio
- To make legacy hardware more like stream processor:
  - Increase computation power
  - Make the best use of available bandwidth
- We study the bandwidth problem

Stream Programming on Legacy Architectures

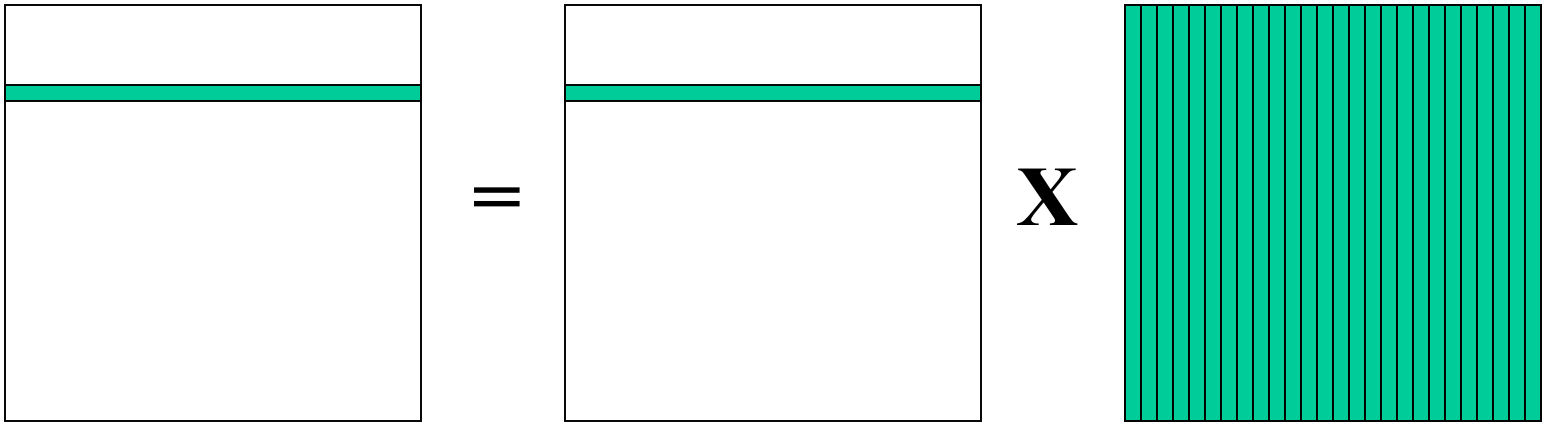
Solutions to Bandwidth Problem

- Reduce the memory bandwidth requirement (algorithm level: strip-mining)
- Re-arrange data layout to more efficiently use bandwidth (SW/HW)
- Overlap memory operations with computations to hide latency (L1/L2 cache as SRF, prefetch, compiler)

MM: Reduce BW Demand

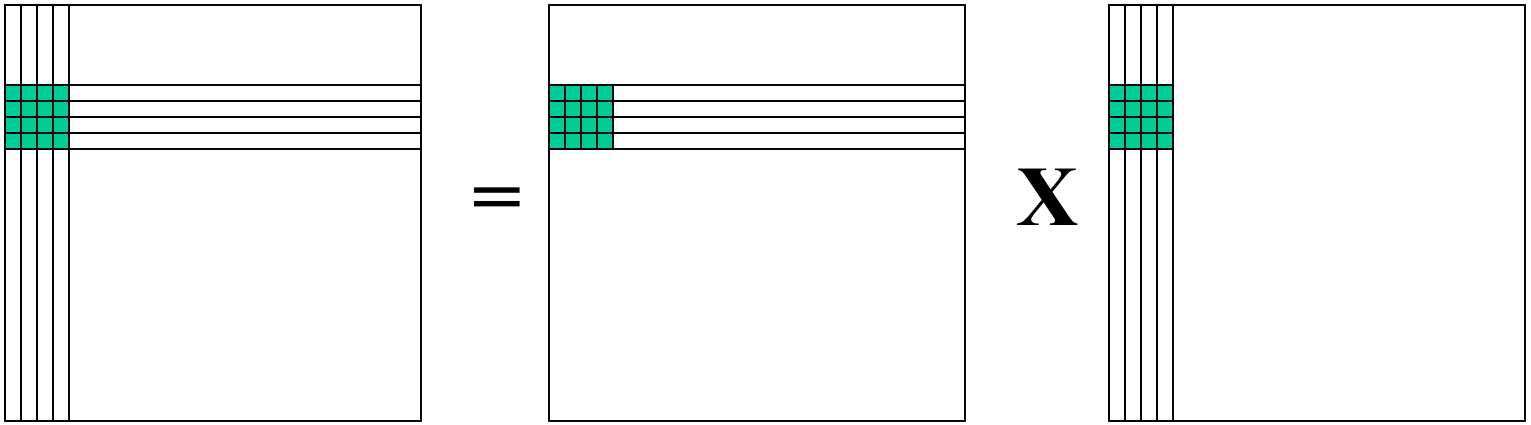
- “2D” strip-mine matrix multiplication (tiling)
- Keep working-set small so it fits in the cache
- Apply as many operations as possible for its life in cache

MM: Naive Way



BW Demand = $O(c_n \cdot n^3)$

MM: 2D Strip-Mined



BW Demand = $O(c_s \cdot n^{2.5})$

$c_s << c_n$, $n \cdot \text{blocksize} < \text{cache size}$

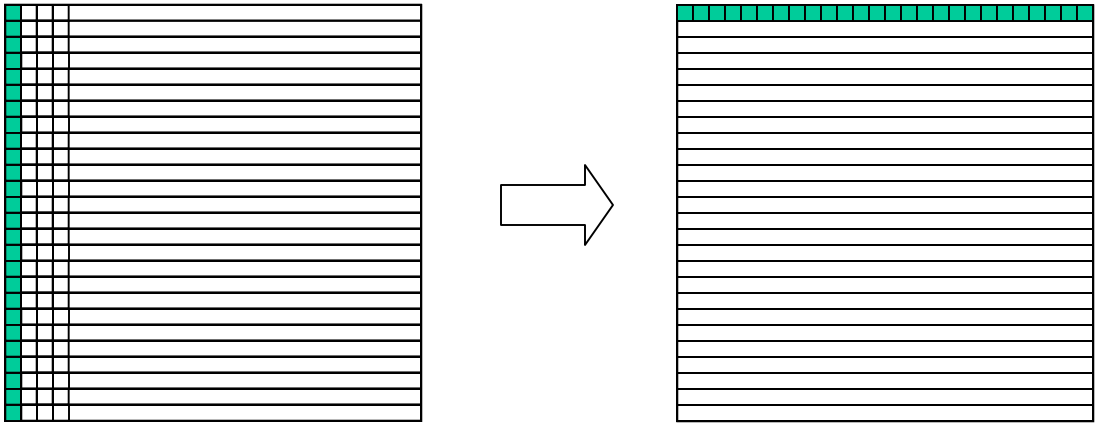
MM: Results

- Pro: strip-mined implementation is a lot faster
- Con: have to rewrite the algorithm
- Problem: cannot “pin” the rows in cache, so they may get evicted
- This may be a general problem for using cache as SRF: not enough explicit control!

Impulse

- Impulse: a memory controller that does gather/scatter
- Can do this in either software or hardware
- Re-organize data layout for better (sequential) access
- At what cost?

MM: Data Re-organization

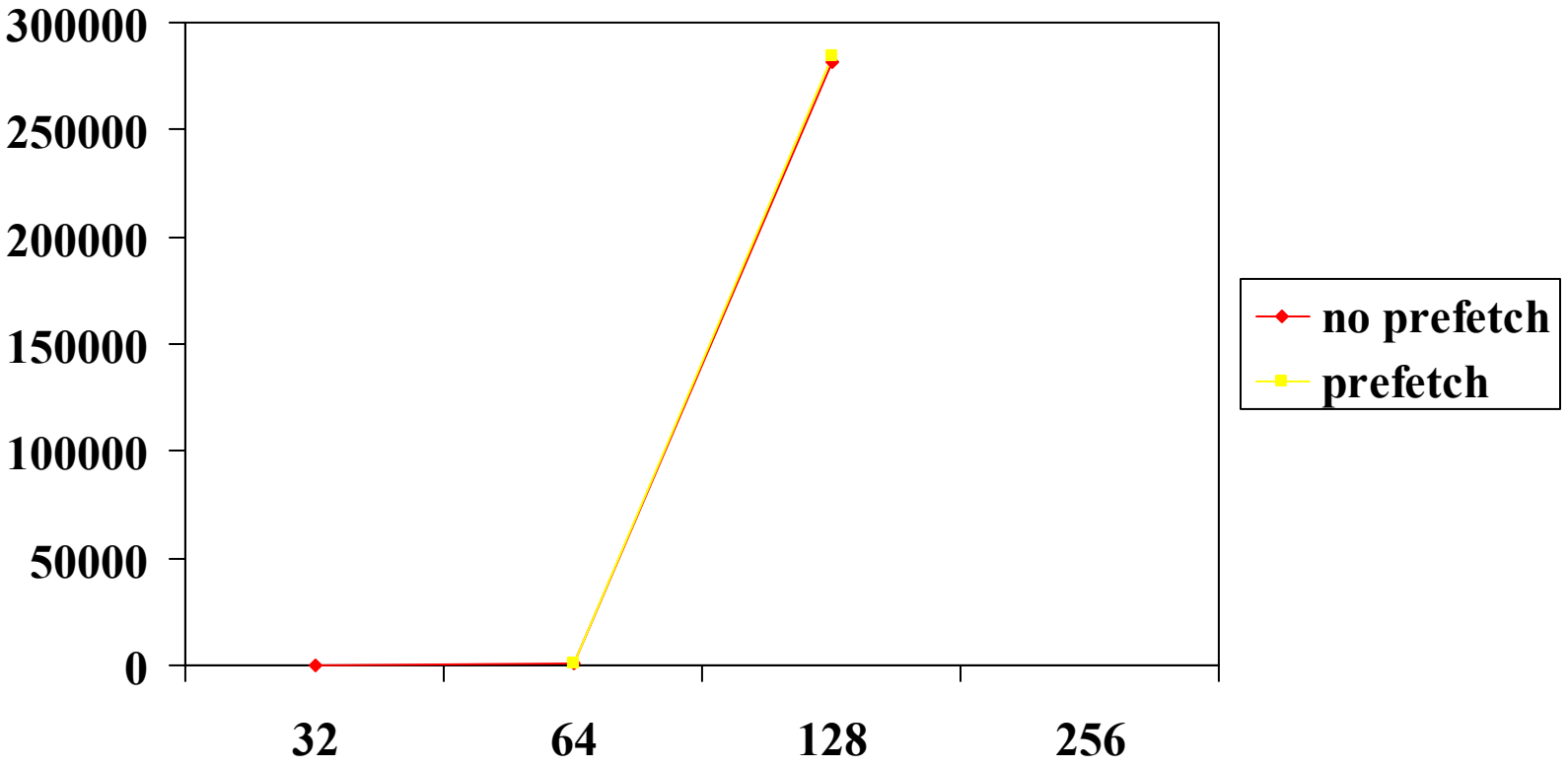


- To access columns in row-major layout, do a strided access
- Can first transpose into column major, and do sequential access – better exploit spatial locality

RSIM configuration

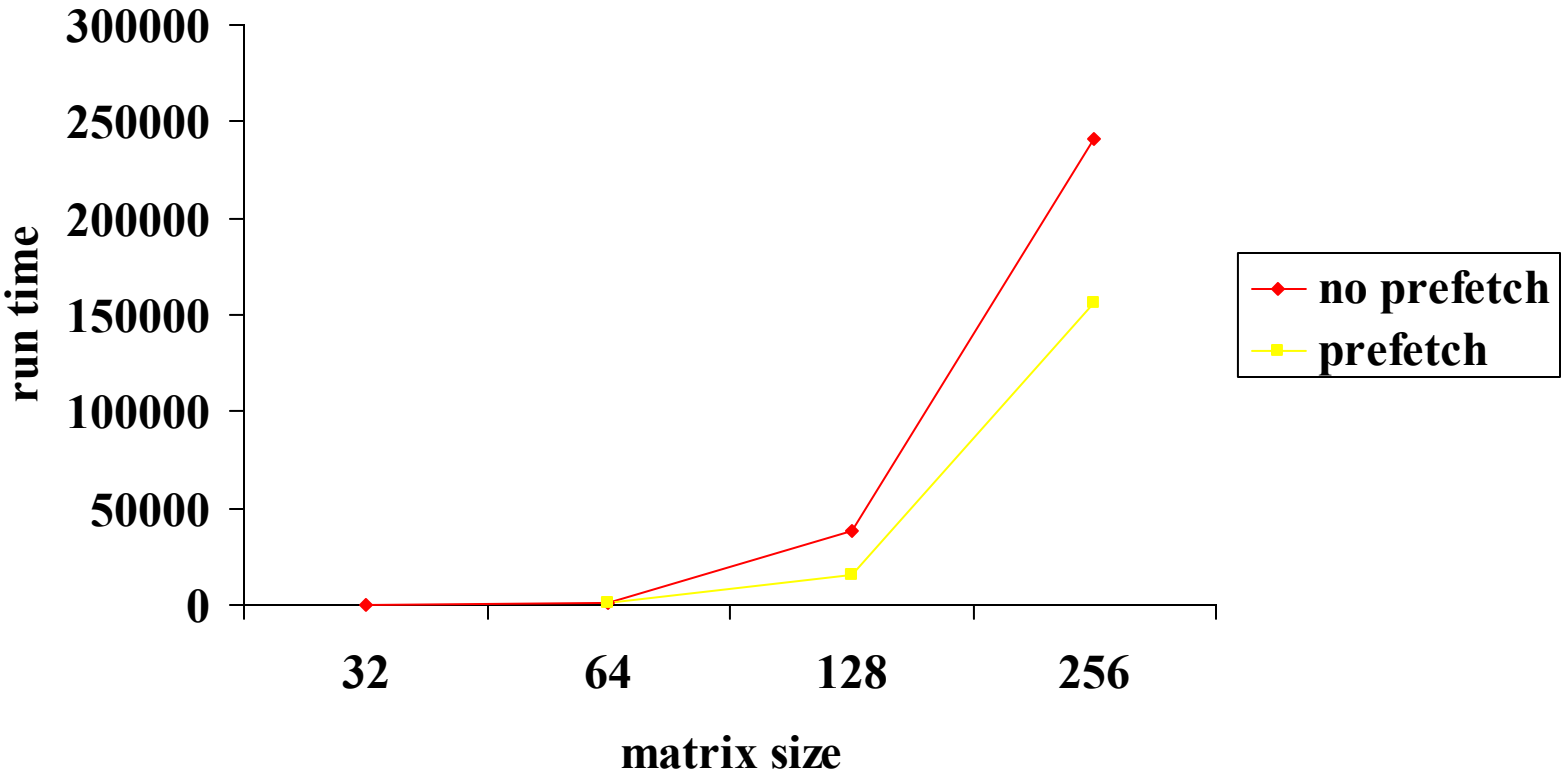
# of processor	1
Processor Speed	1.2GHz
Issue Width	4
Instruction window size	64
Memory queue size	32
Cache line size	32 bytes
L1 cache size	16KB
L1 latency	1 cycle
L2 cache size	64KB
L2 latency	5 cycles
Memory Latency (for L2 miss)	72 cycles

Matrix Multiplication (row, col)



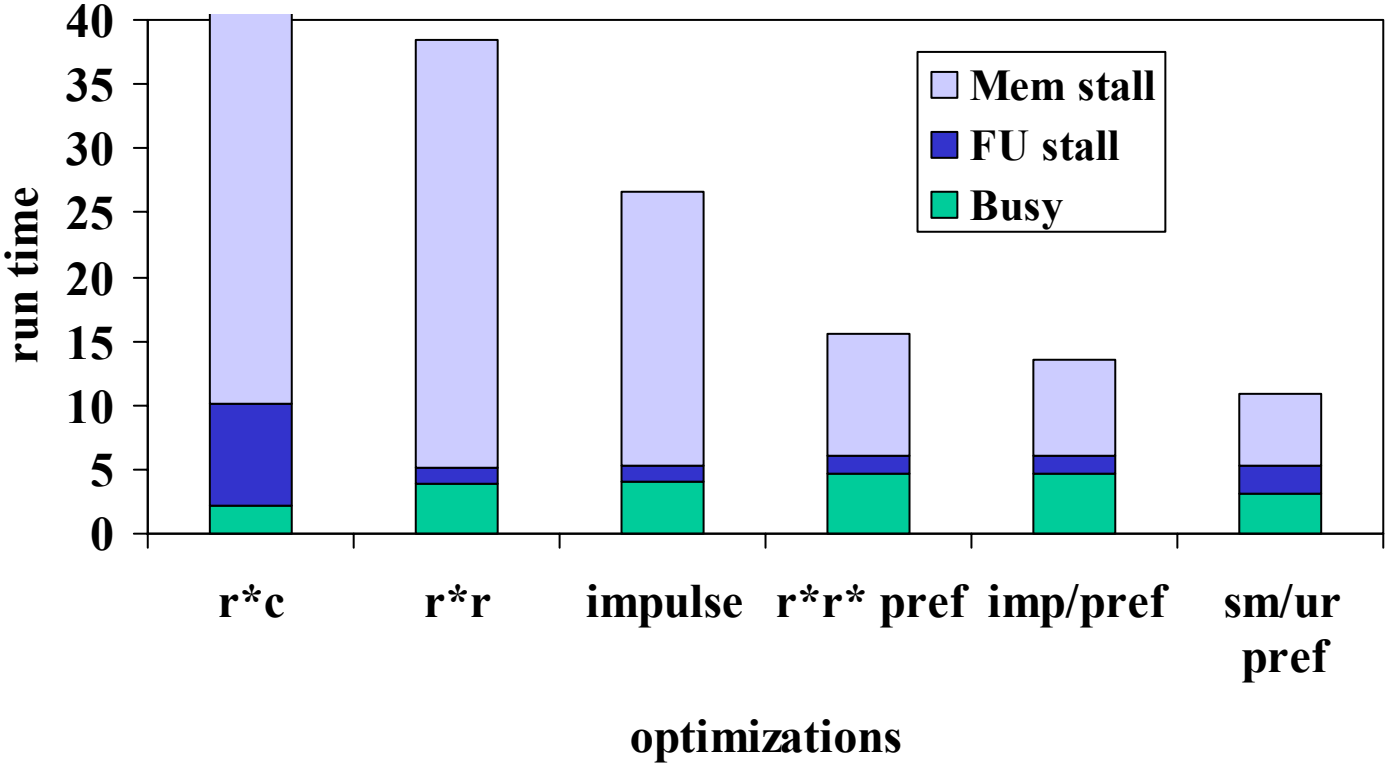
- computation increases N^3
- no benefit from prefetch (degrades performance slightly)
- most of prefetch classified as unnecessary

Matrix Multiplication II (row X row)



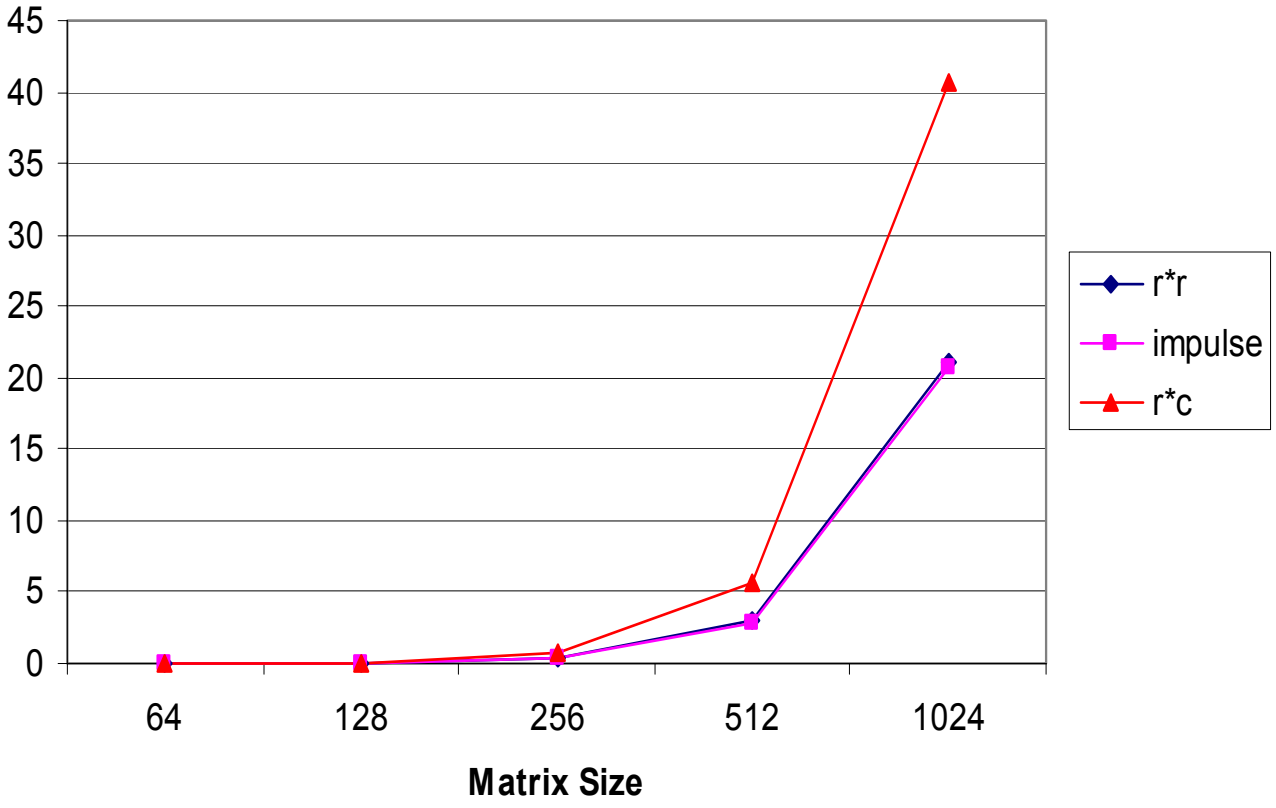
- compiler inserted prefetch
- significant increase in performance as size increase beyond cache size

Performance of Matrix Mult (128)



- why is impulse better then r*r? higher L2 cache hit rate?
- by performing both optimization, memory stall time reduced significantly
- (r*c not drawn to scale)

Performance on Saga



Compiler Optimizations

- From last week's status update:
  - Treat streams as arrays of records
  - Perform data reuse analysis to identify probably cache misses
  - Use software pipelining to prefetch records in advance (split loop into prolog, steady state, epilog)

MediaBench results

- Performance is worse with prefetch
- Plausible explanations:
  - These apps stream off files on disks, not arrays resident in memory
  - Redundant prefetches add overhead (prefetch in inner loops w/o unrolling?)
  - Mismatch between gcc assumption of hardware and simulated hardware
- Possible solutions:
  - Rewrite the apps
  - Fix compiler/simulation environment

Problems

- "Short Stream Effects"
  - Small steady state loops
  - Prefetches from prolog are 'late'
  - No stream scheduling
- Difficult to debug
  - GCC has many hacks and poor docs
  - Bugs in reuse analysis code result in 'unnecessary' prefetches
  - Should have used SUIF to emit C then GCC

Problems (cont.)

- Hardware Stride Prediction
  - For uniprocessors w/affine array accesses, hardware is pretty good
  - Problems for shared memory/MP
  - Added logic/area
  - We can handle indexed streams B[A[i]]

Conclusions Future Work

- Use L1 as SRF, L2 as buffer
- Unroll outer loops, fuse inner loops to increase computation/reduce prolog effects
- Ideally: use cache partitioning and separate prefetch thread to handle stream scheduling
  - We don't *really* utilize knowledge of access patterns early enough

Extra slides follow

Cache as SRF

- Prefetch data given knowledge of memory access pattern
- Either by explicit use of prefetch instruction or by implicit touch of data
- Prefetch requires loop-unrolling to reduce overhead and redundancy
- Prefetch controls what and when gets into the cache; what about what gets evicted?

Cache as SRF (cont'd)

- L1 or L2?
- Ideally, do computation with data in L1 and overlap that with prefetch into L2
- L1 seems more useful as SRF (needs to combine with strip-mining to keep data-set really small?); L2 kind of redundant
- Unless we do hierarchical strip-mining (haven't tried yet)