Modeling Cache Sharing for MPI Programs on Multi-core Machines

Bin Bao, Chen Ding
University of Rochester

Nov 10, 2011

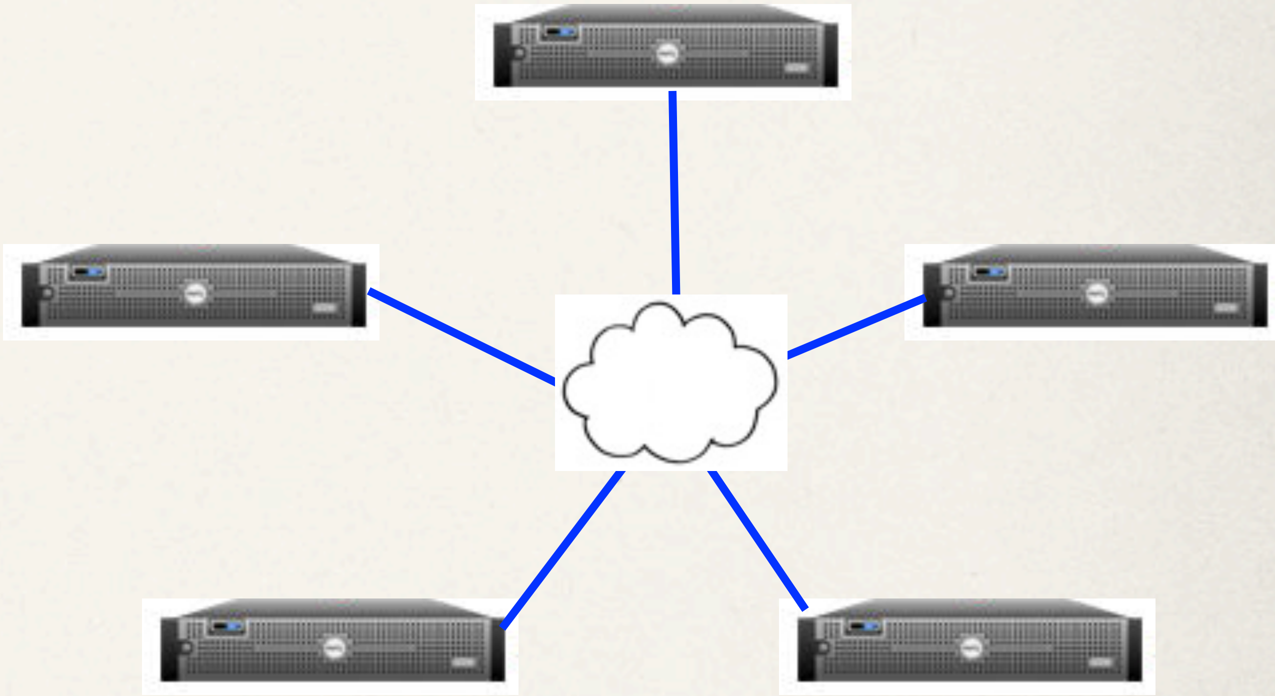
The 10th Workshop On Compiler-Driven Performance

Multi-core Popularity

- More and more cluster machines are using multi-core processors
- TOP500.org (June 2011):
    - "Quad-core processors are used in 46.2 percent of the systems, while already 42.4 percent of the systems use processors with six or more cores."

Programming on Cluster

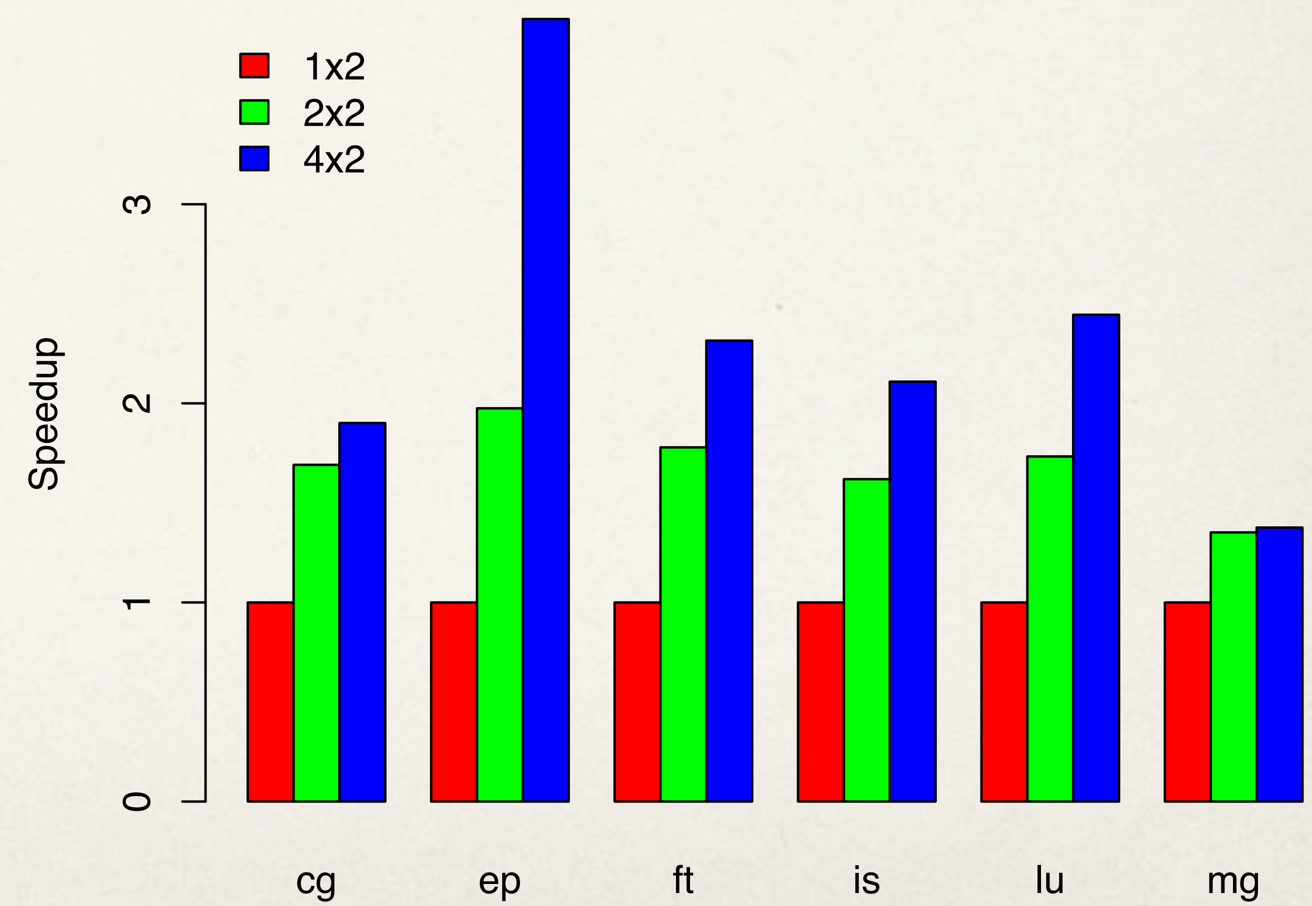
- MPI (Message-Passing Interface) is still dominant
- Scalability issues, e.g.
  - Load balance
  - Communication overhead
  - Multicore: resource contention



Performance Impact of Resource Sharing

- Experimental studies: Chai et al. [CCGRID'07], Saini et al. [SC'09], etc.

- Intel Nehalem E5520 (4 cores) (Shared 8MB L3 cache)
- GCC 4.4.1, MPICH2 1.4.1



Goal: Modeling Cache Contention

- Tool: reuse distance (aka LRU stack distance), the number of distinct data elements accessed between two consecutive references to the same element

a b c c d a

|← rd=3 →|

- Reuse distance can be used to calculate cache miss rate
  - Program A's cache miss rate = P(A's reuse distance ≥ cache size)

Locality (Reuse Distance) Scaling

- Strong scaling: fixed total problem size
- Fixed-distance reuses and scaled-distance reuses

$$\begin{matrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ & & \dots & \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{matrix} \; X \; \begin{matrix} b_1 \\ b_2 \\ \dots \\ b_n \end{matrix} = \begin{matrix} c_1 \\ c_2 \\ \dots \\ c_n \end{matrix}$$

Locality (Reuse Distance) Scaling

- Strong scaling: fixed total problem size
- Fixed-distance reuses and scaled-distance reuses

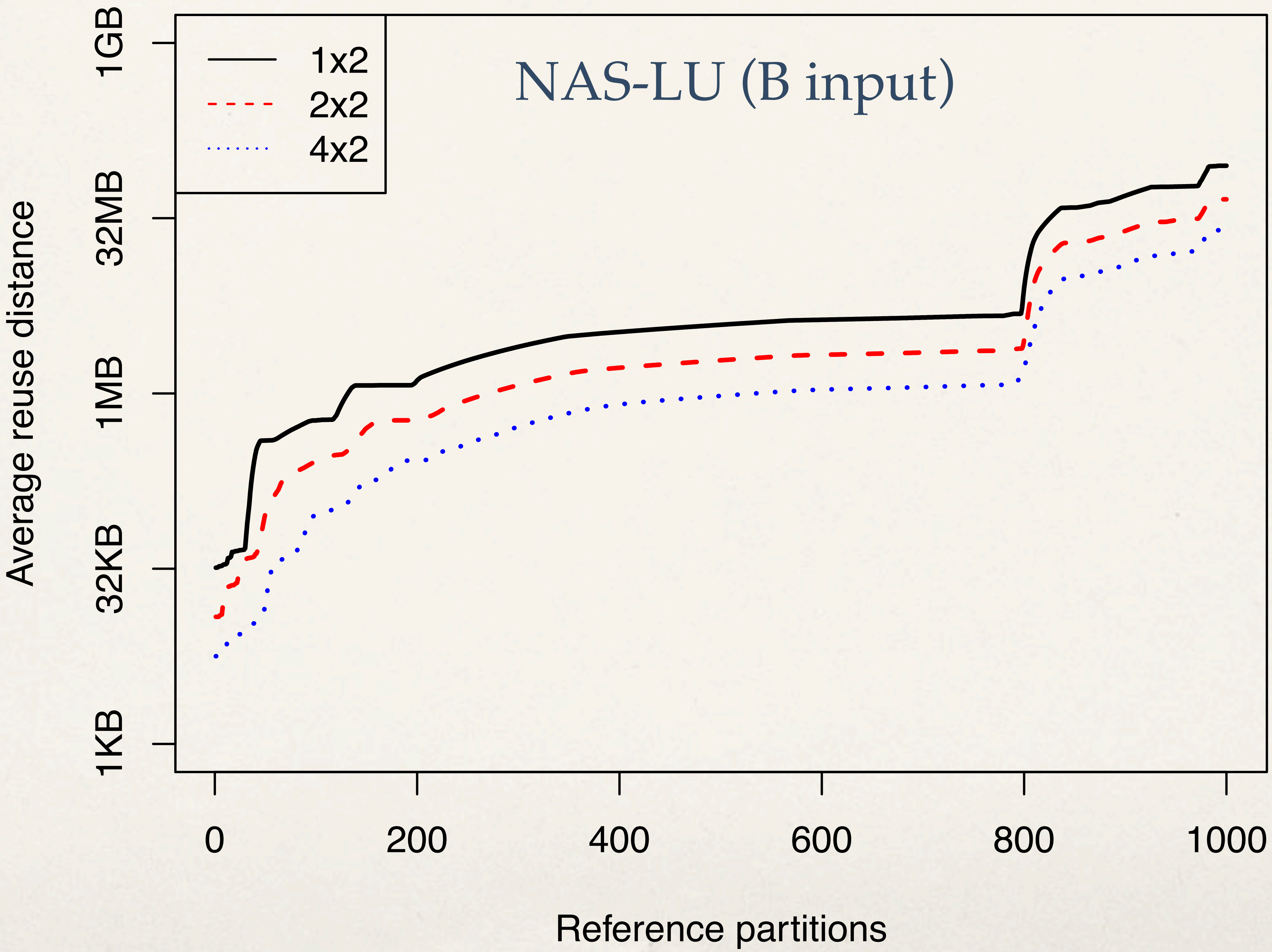
$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ & \dots & & \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{bmatrix}$$

Locality (Reuse Distance) Scaling

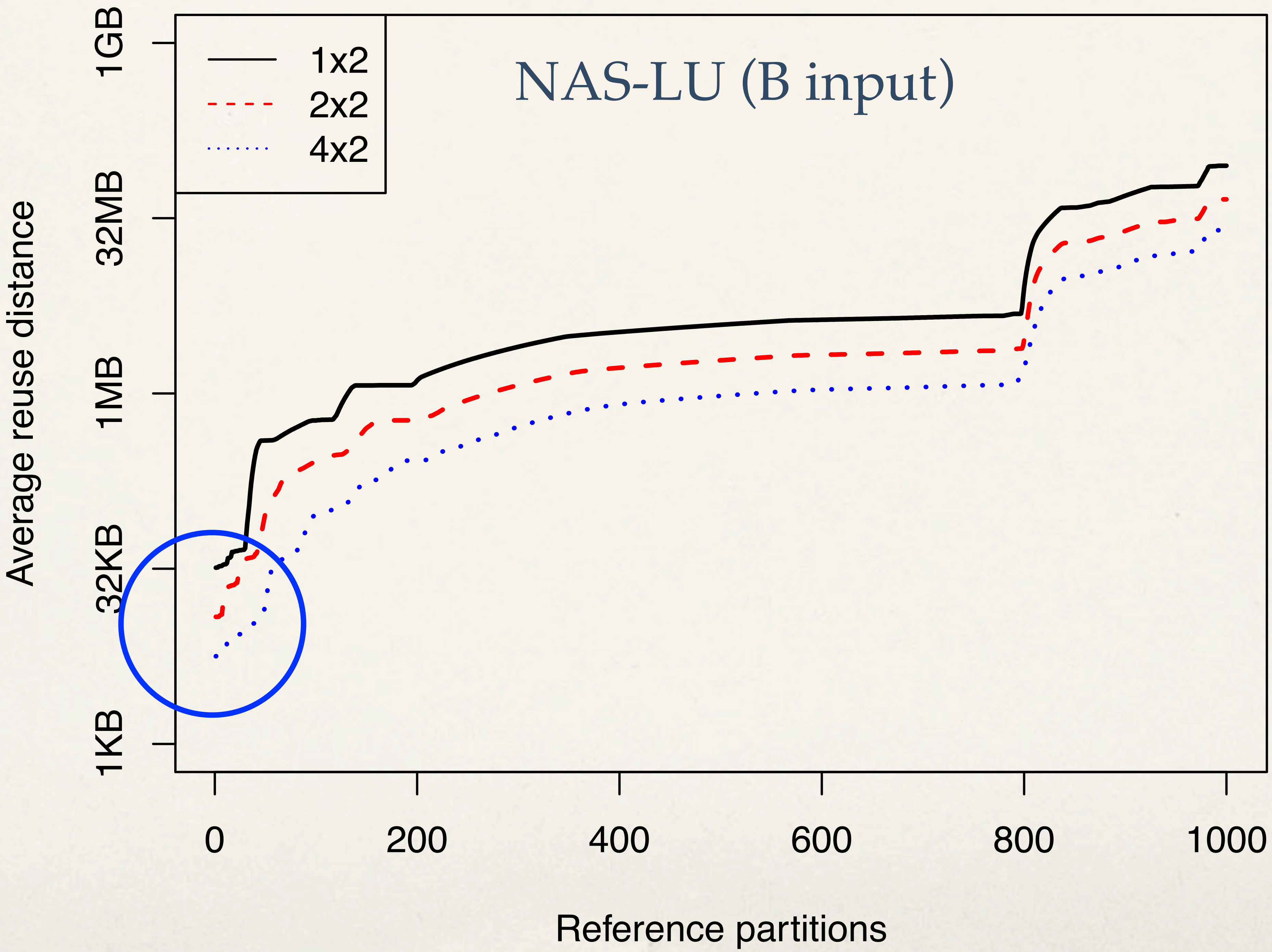
- Strong scaling: fixed total problem size
- Fixed-distance reuses and scaled-distance reuses

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ & & \dots & \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} X \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{pmatrix}$$

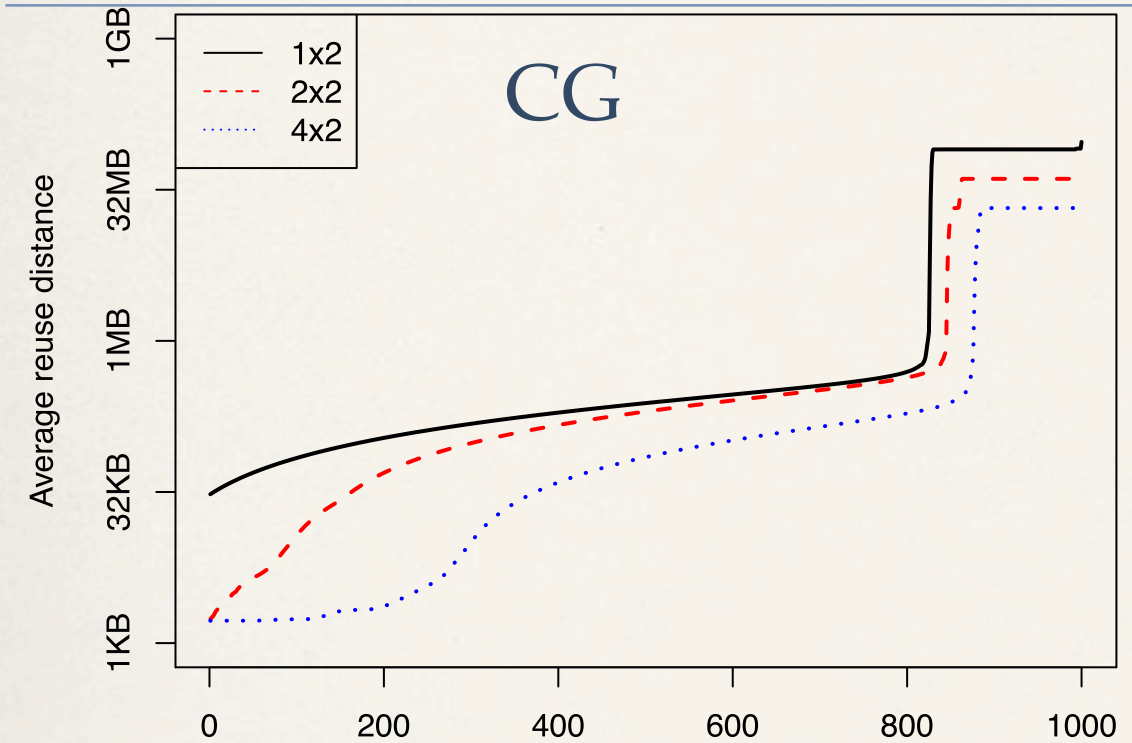
Reuse Distance Reference Histogram

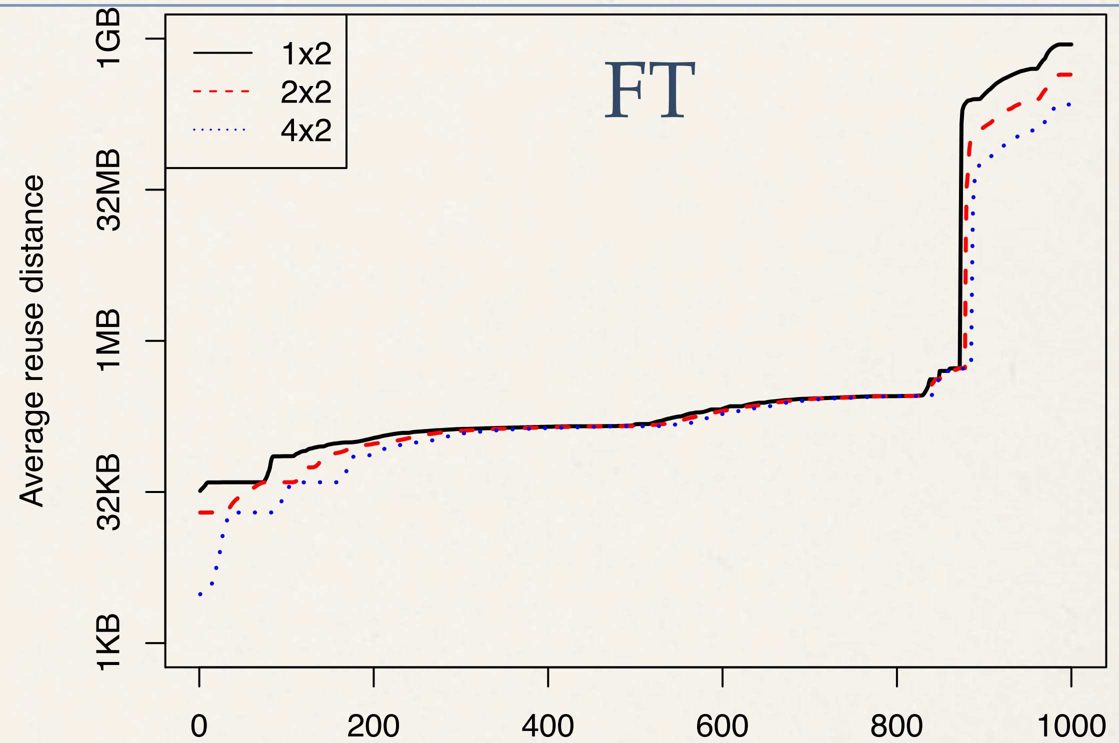


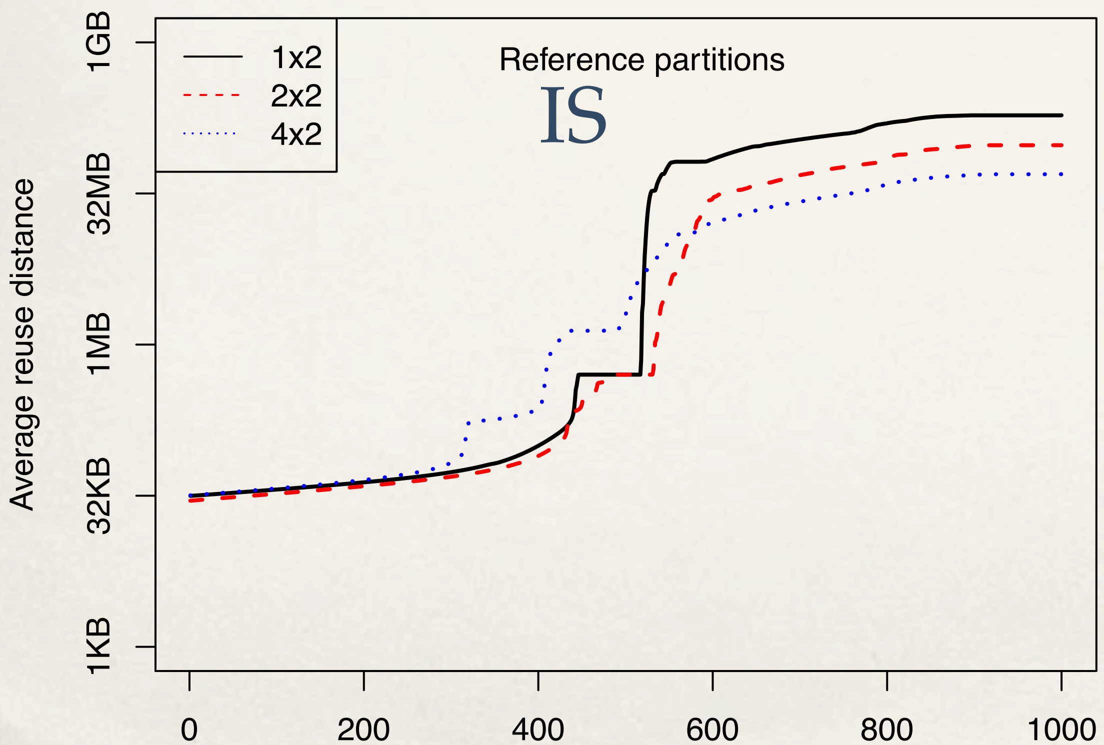
Reuse Distance Reference Histogram

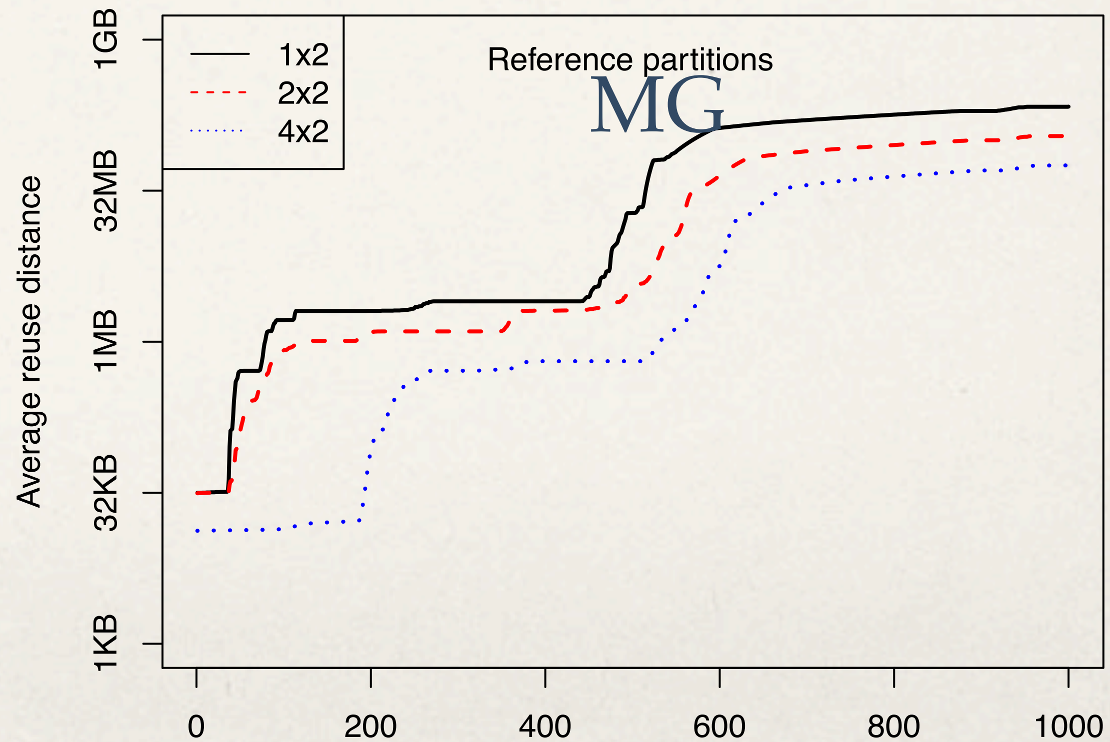


More Examples









Linear Regression Based Reuse Distance Prediction

- For partition i:

$$rd_i = a_i \times (1/nproc) + b_i$$

- The model captures scaled-distance reuses and fixed-distance reuses
- Each partition is independent

Cache Sharing

- General dilation model [Xiang et al. PPoPP'11]

rd = 5

Program A: a b c d e f a

ft = 4

Program B: k m m m n o n

rd' = rd+ft = 9

Program A&B: akbcmdmemfnona

- Symmetric MPI programs: uniform interleaving assumption

rd = 5

Task A: a b c d e f a

rd' = 5

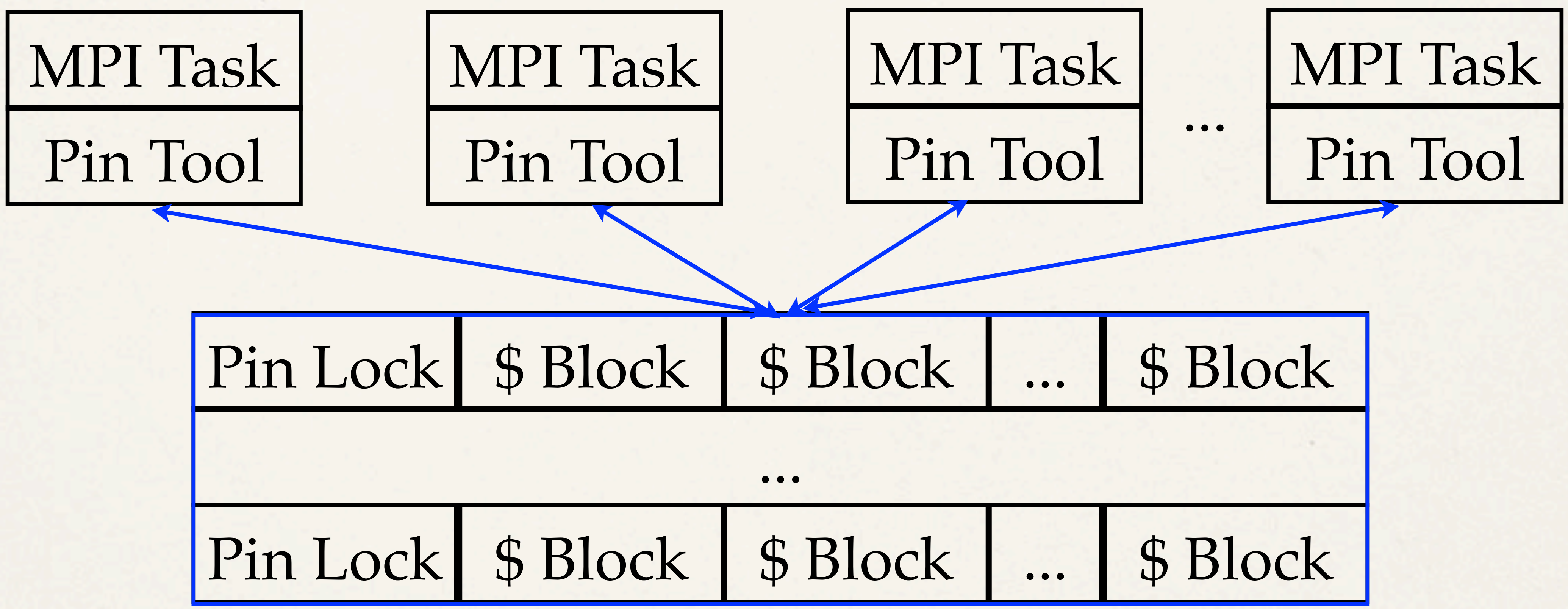
Task B: k l m n o p k

rd'' = 11

Task A&B: abcklmdenofpak

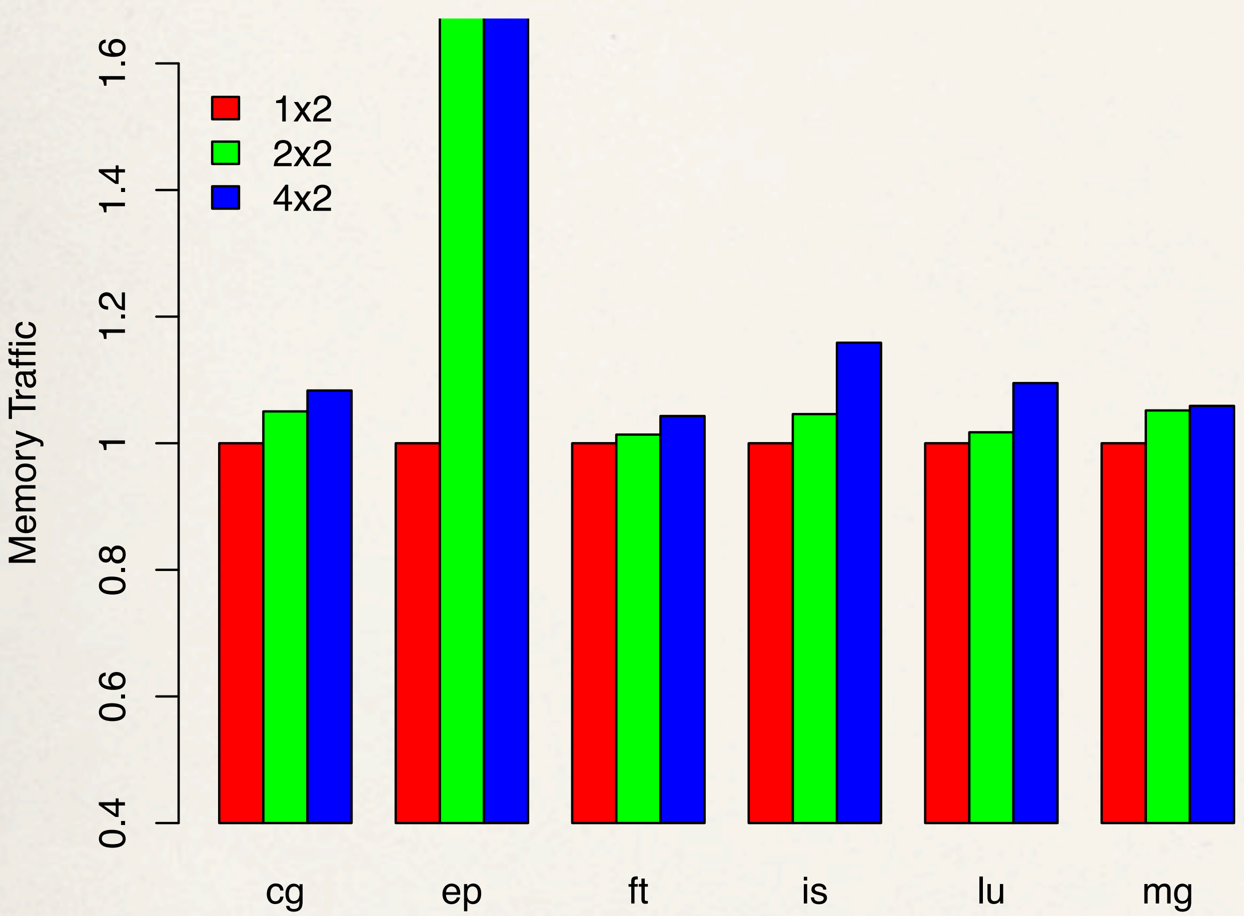
Experiments

- Pin-based trace cache simulator (16-way LRU, 8MB)

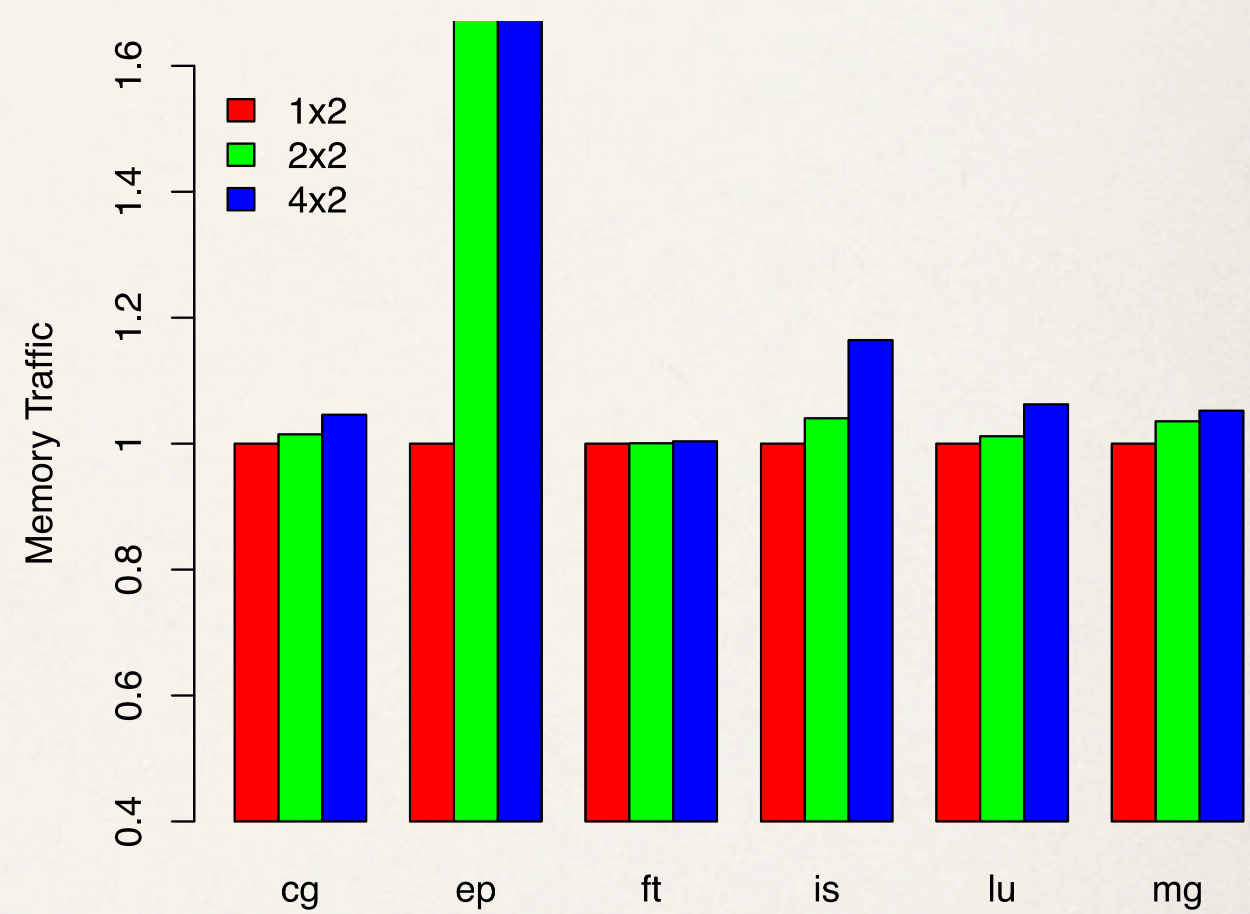


- Performance counters (OProfile, LLC Misses)

Cache Simulator vs Reuse Distance Based Calculation

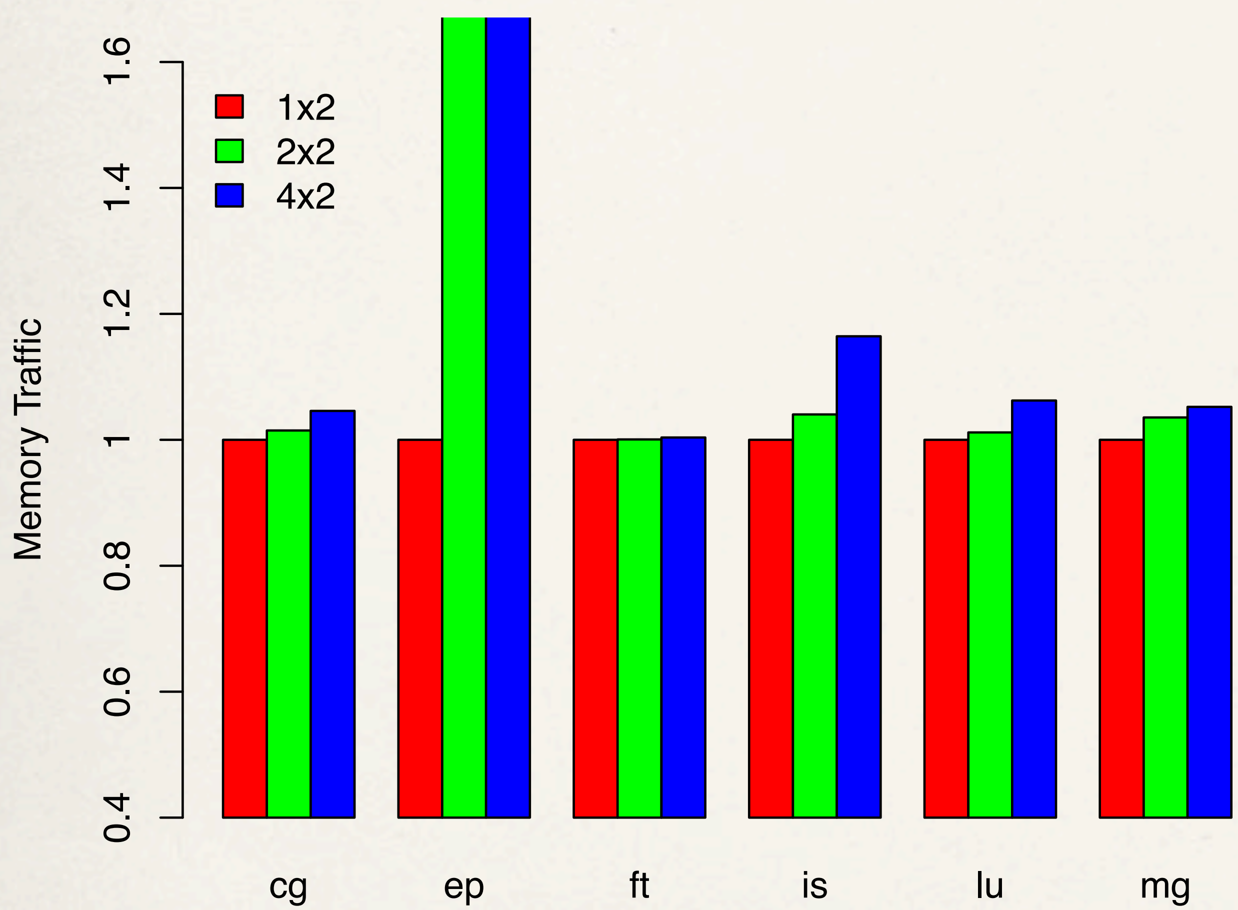


(a) Cache Simulator

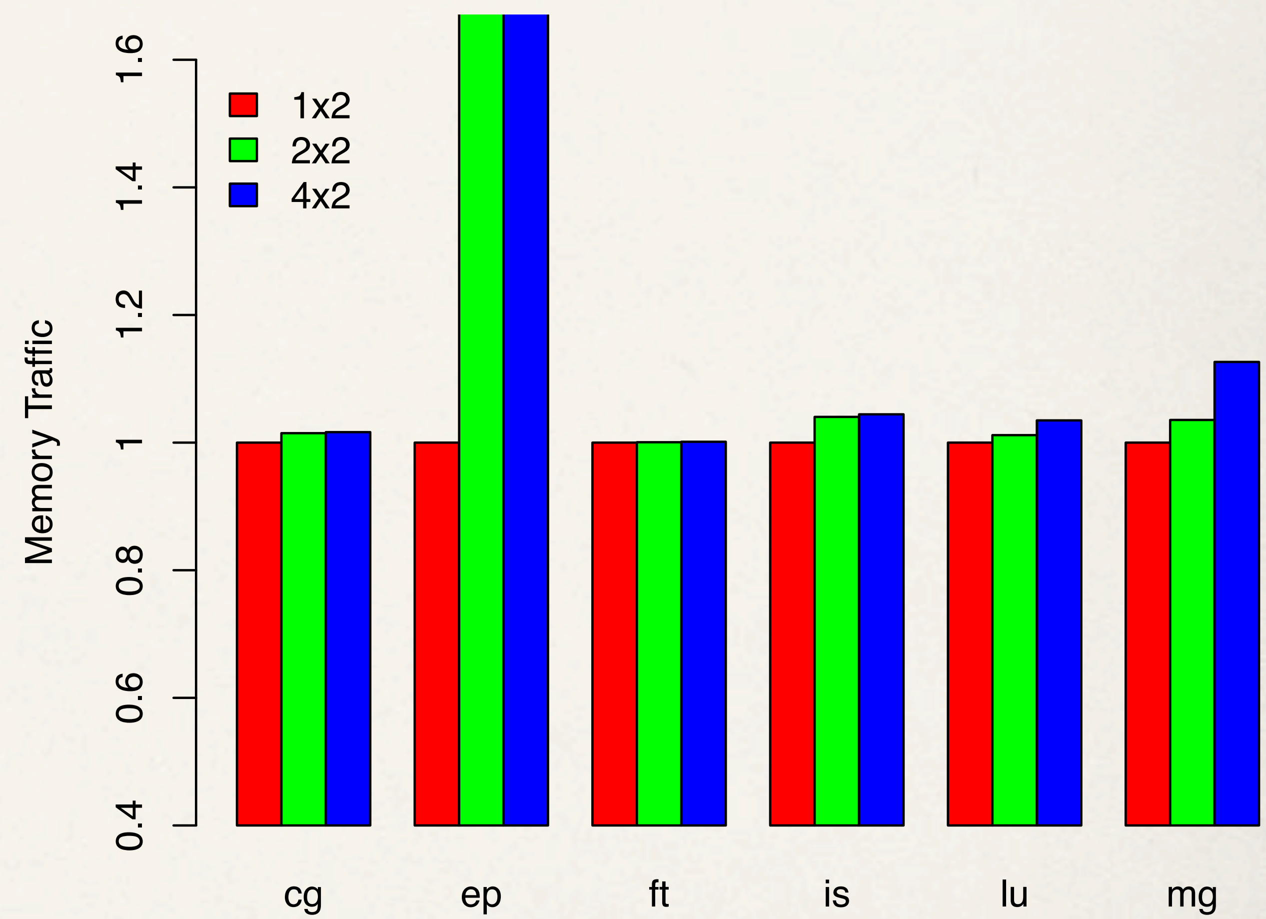


(b) Reuse Distance Based Calculation

Reuse Distance Prediction

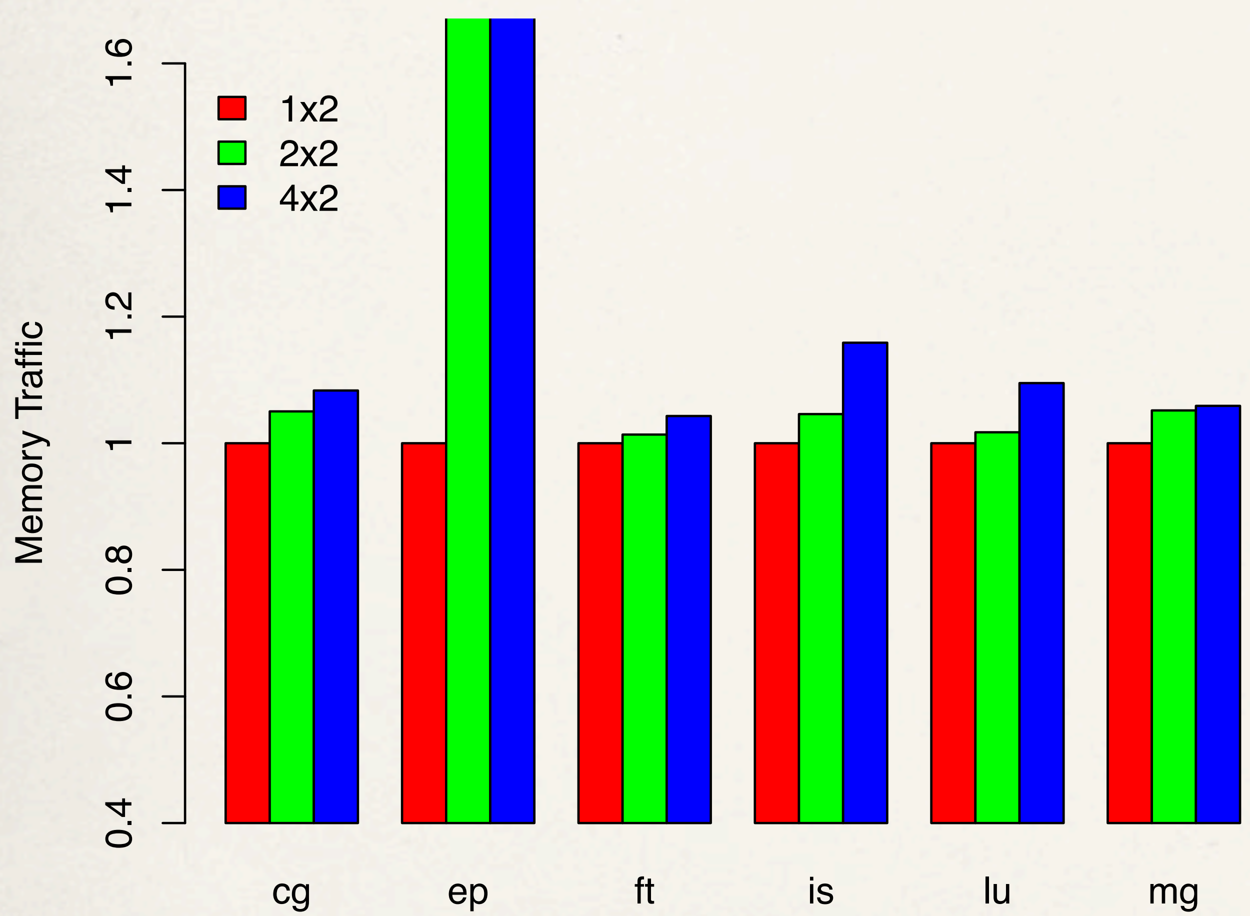


(a) Reuse Distance Based Calculation

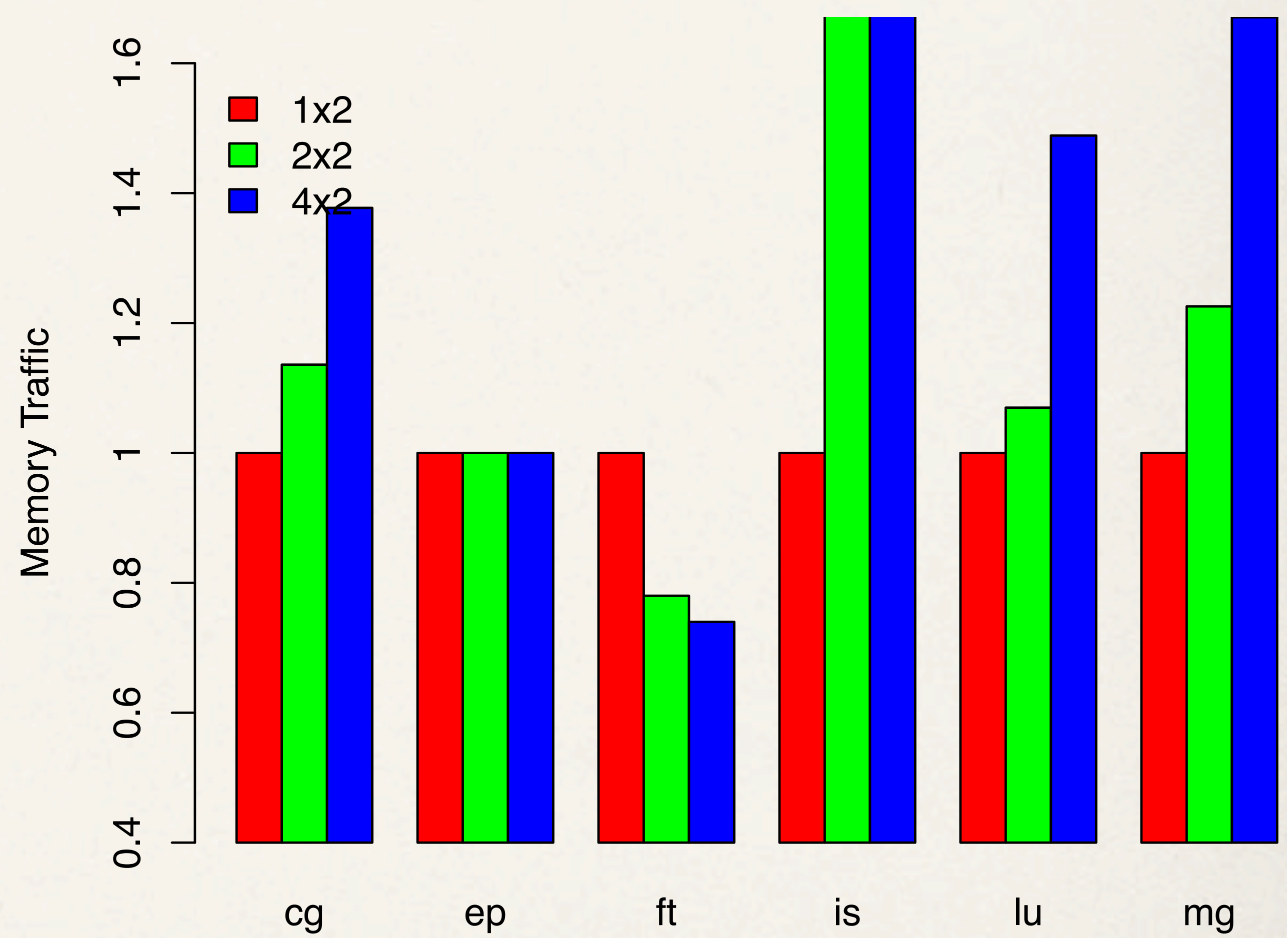


(b) Reuse Distance Prediction (8-task)

Hardware Performance Counter

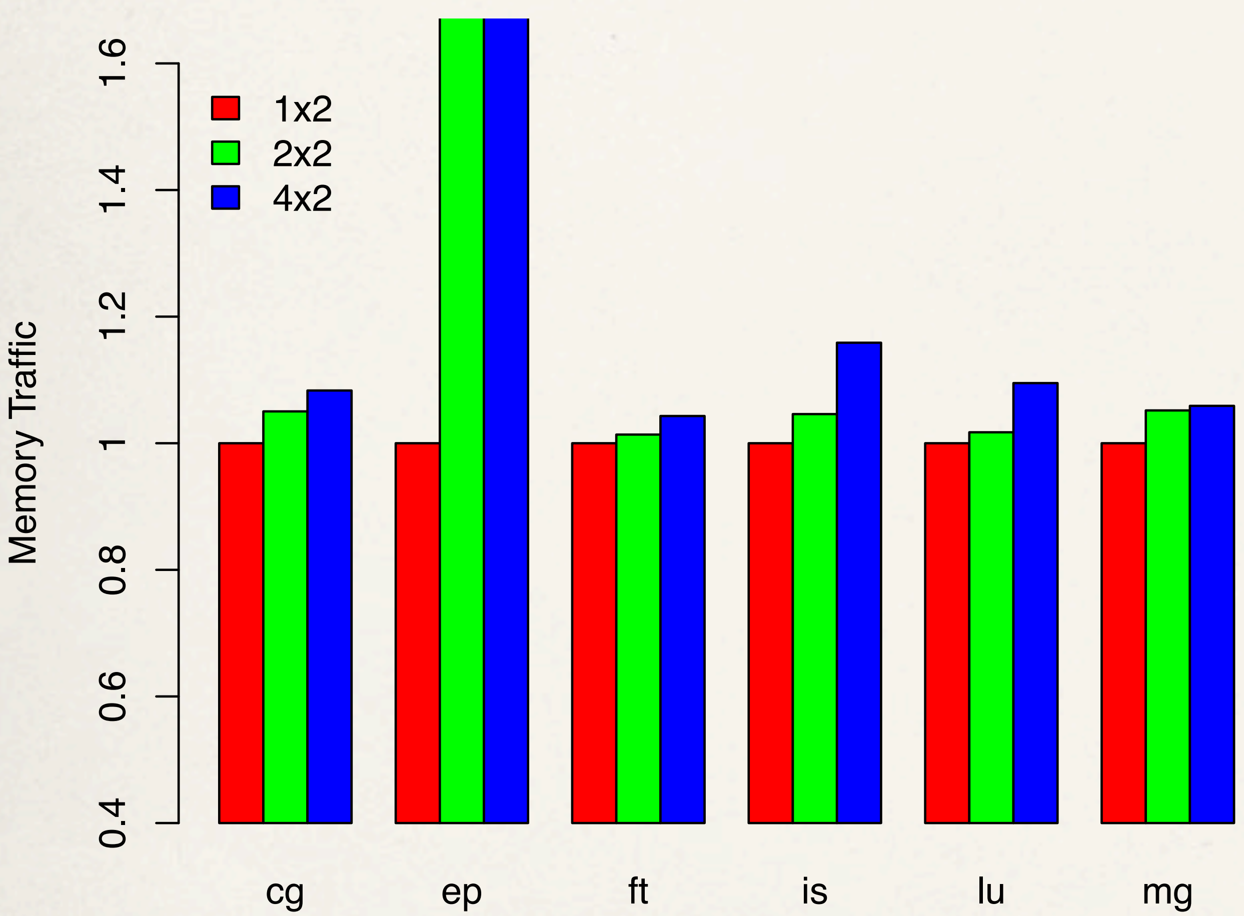


(a) Cache Simulator

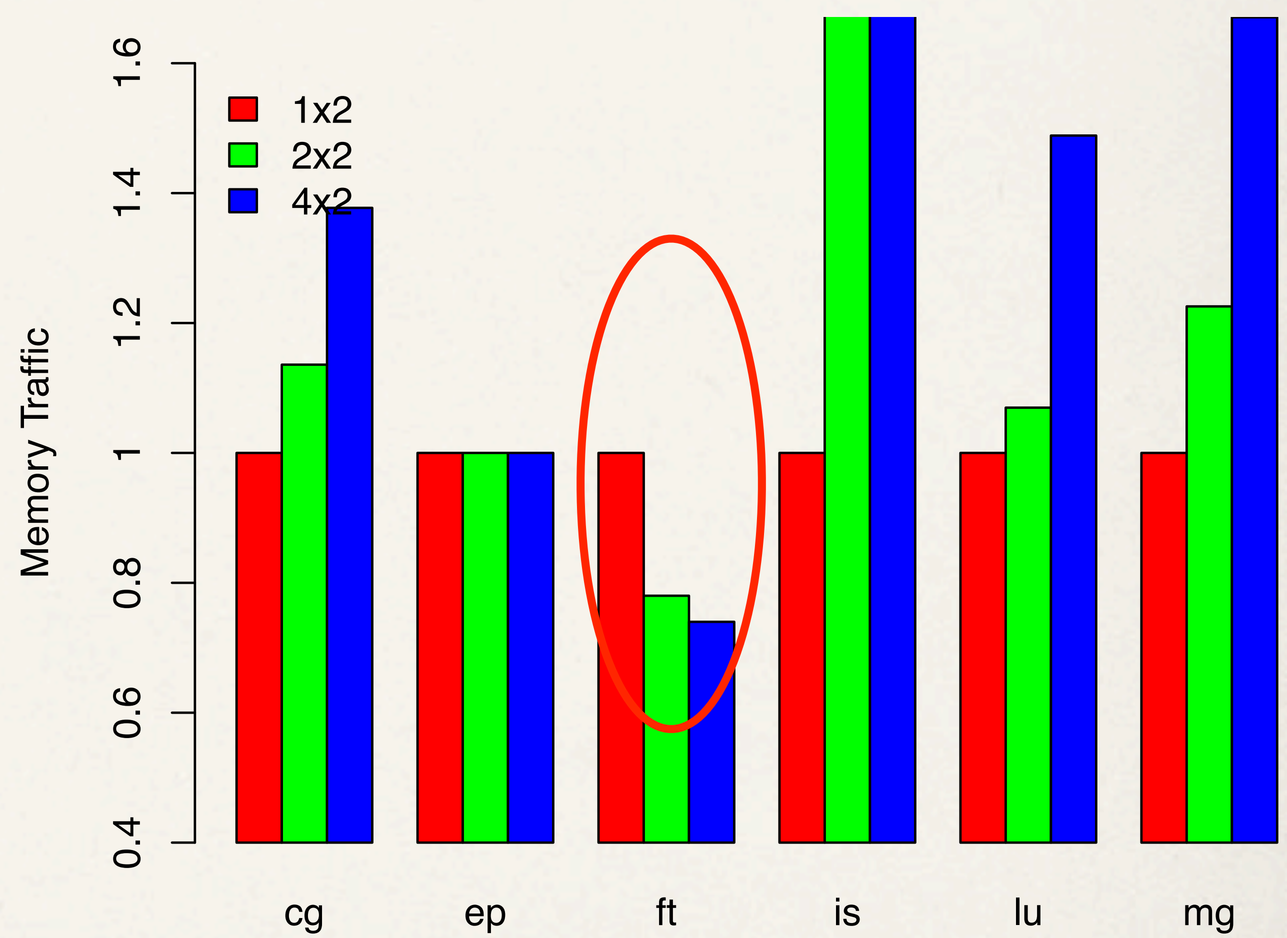


(b) Performance Counter

Hardware Performance Counter

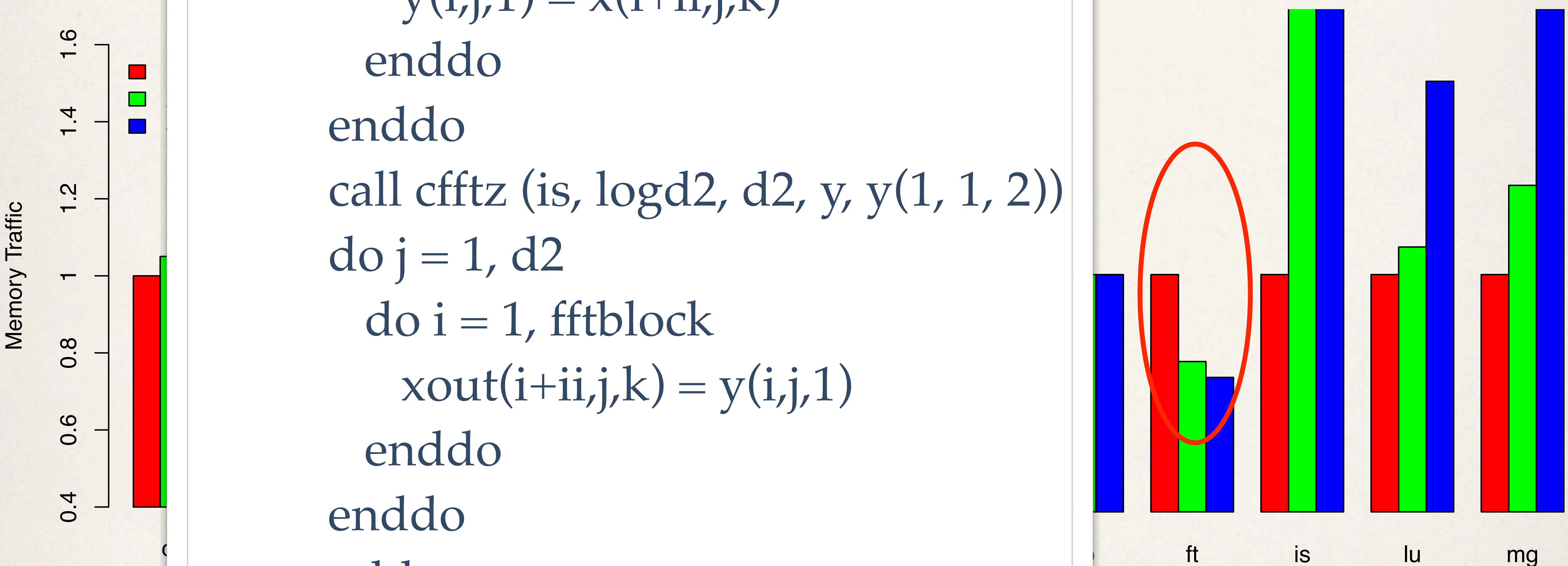


(a) Cache Simulator



(b) Performance Counter

Har… Counter




```
do k = 1, d3
 do ii = 0, d1 - fftblock, fftblock
   do j = 1, d2
     do i = 1, fftblock
       y(i,j,1) = x(i+ii,j,k)
     enddo
   enddo
   call cfftz (is, logd2, d2, y, y(1, 1, 2))
   do j = 1, d2
     do i = 1, fftblock
       xout(i+ii,j,k) = y(i,j,1)
     enddo
   enddo
 enddo
enddo
```

…erformance Counter

Summary & Future Work

- Reuse distance reference histograms show clear patterns
  - Linear regression based reuse distance prediction
- Coarse-granularity uniform interleaving assumption
- Verified with a Pin-based cache simulator
- Memory bandwidth contention modeling