# ThreadSpotter

John Hollis
Sr. Director Consulting Services, Worldwide
Rogue Wave Software

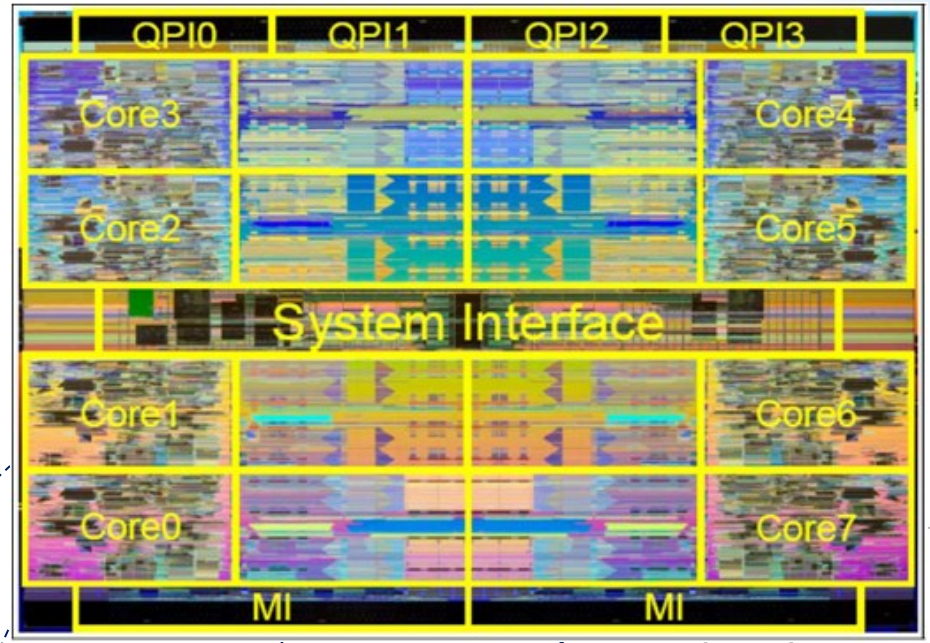
# Agenda

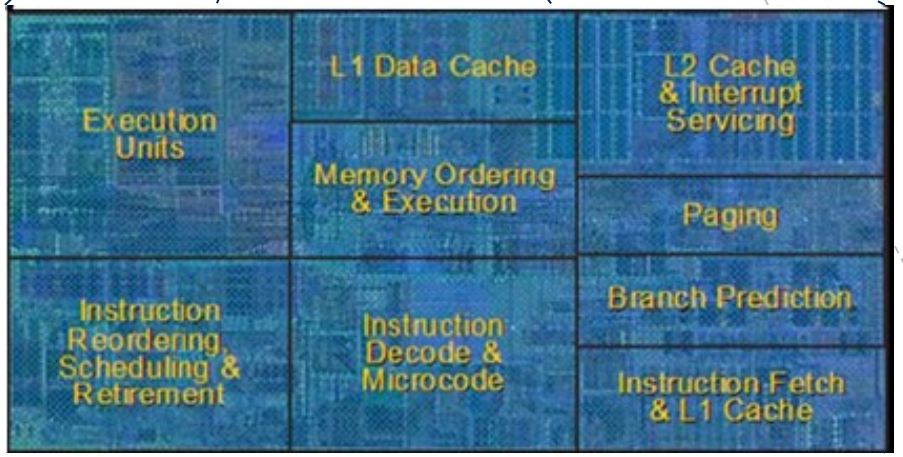
- Some Remarks about Caching
- What Can Go Wrong?
  - Partially used structure
  - Alignment problem
  - Inefficient loop nesting
  - Non-temporal data
  - False sharing
- ThreadSpotter Settings
- Sampling
- Creating Reports
- Questions

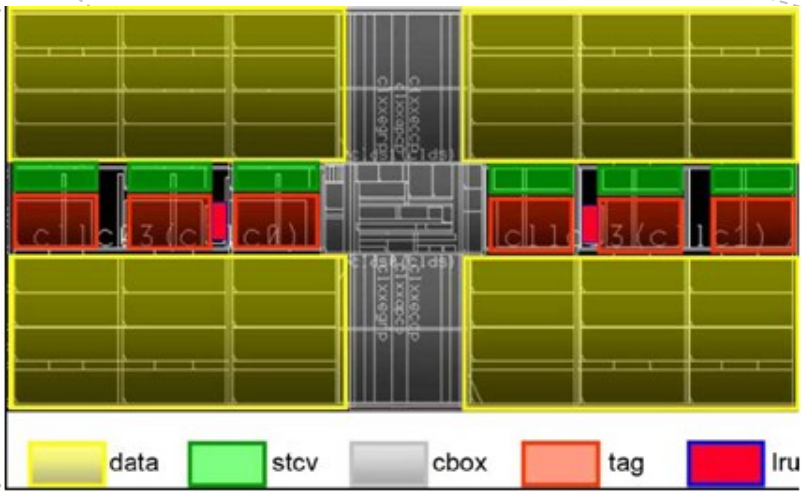


Copyright 2012 Rogue Wave Software, Inc.

# How is the silicon used (i7-Ex)?



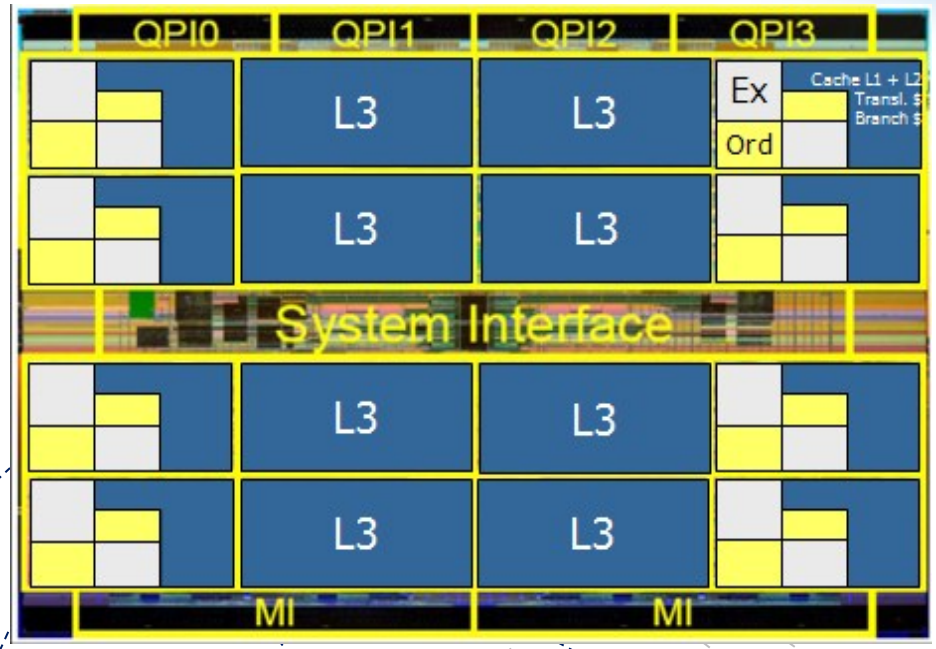



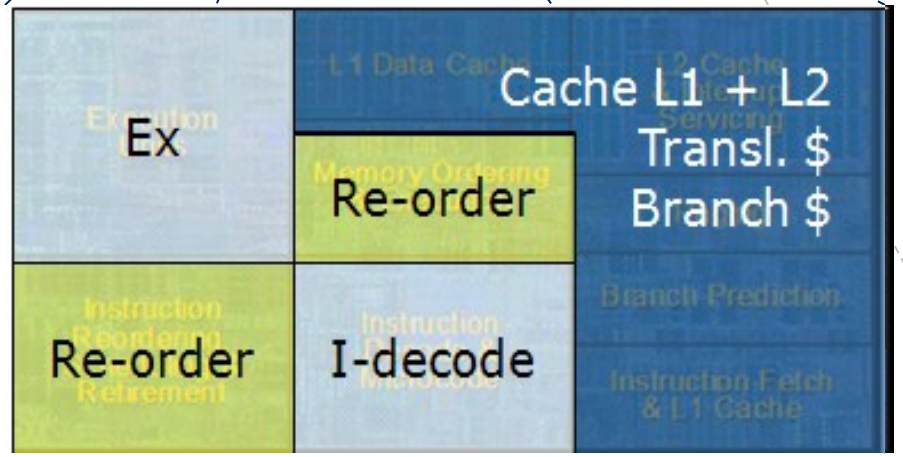



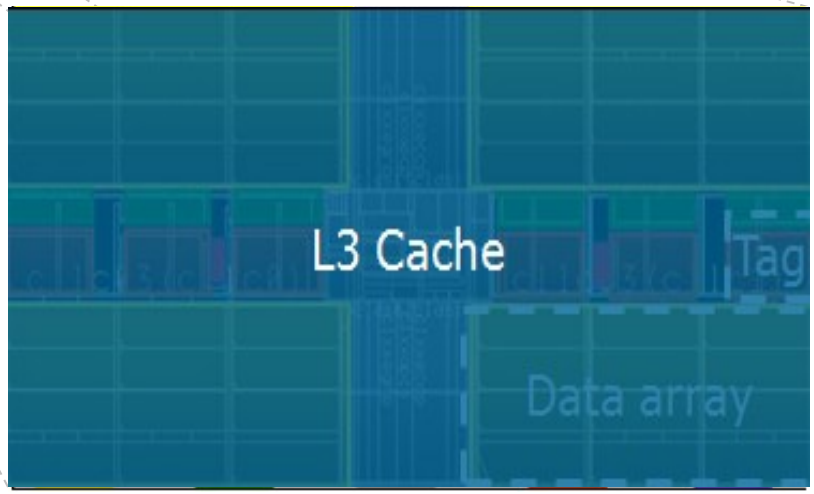


Copyright 2012 Rogue Wave Software, Inc.

# How is the silicon used (i7-Ex)?



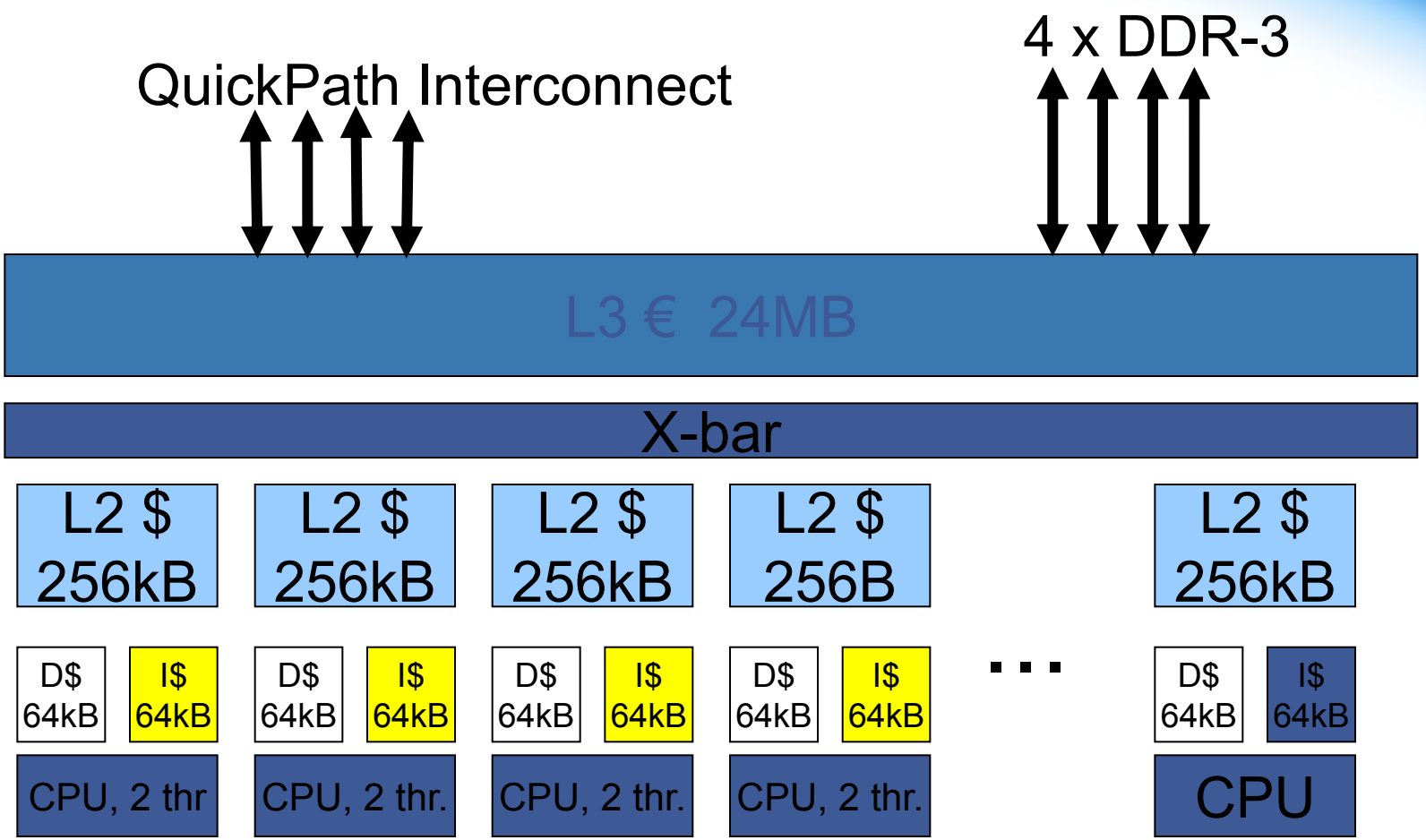





Copyright 2012 Rogue Wave Software, Inc.

# Intel i7





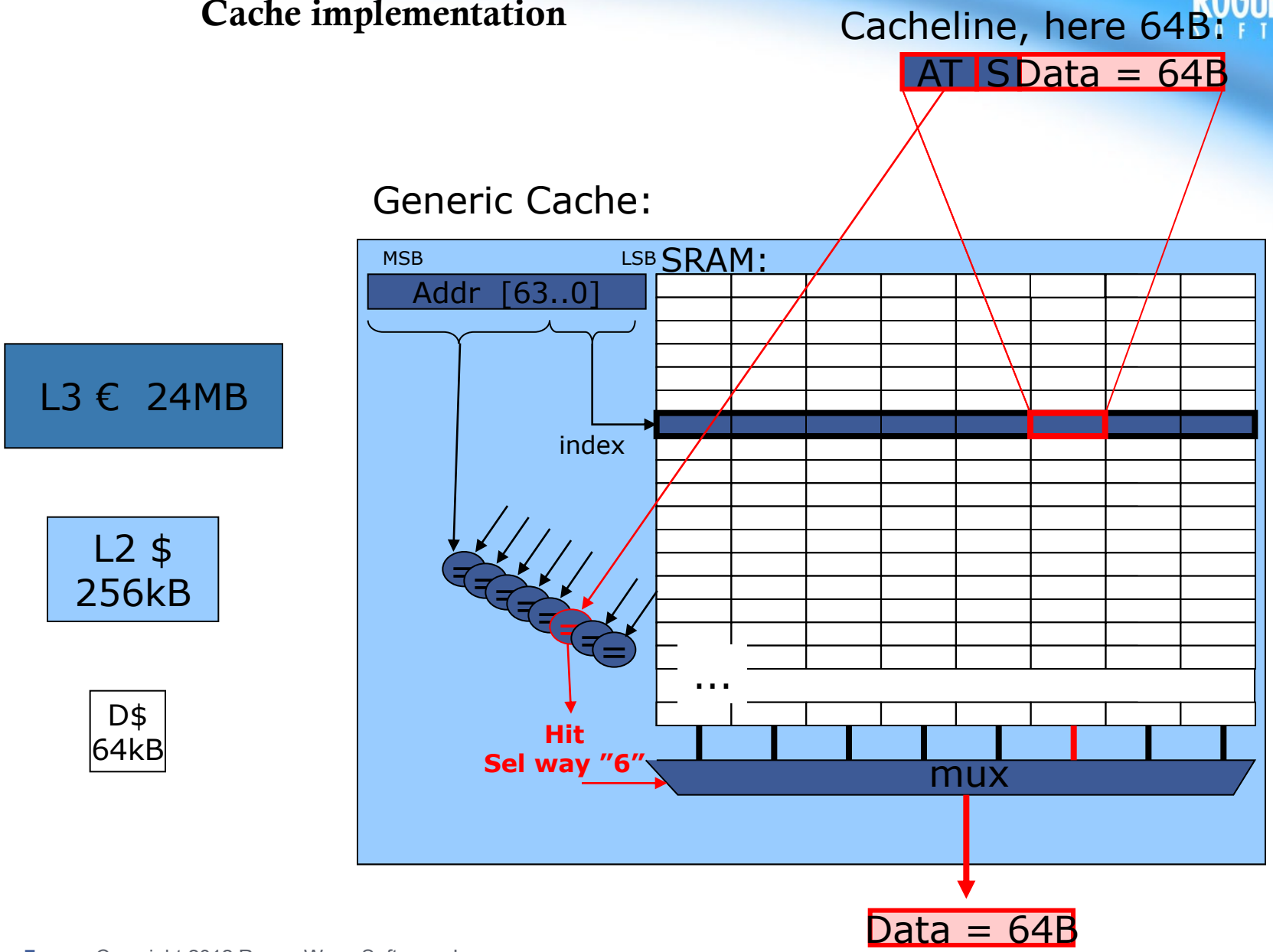
Copyright 2012 Rogue Wave Software, Inc.

# What is the potential gain of optimizing cache usage?

- Latency difference L1$ and mem: ~50x
- Bandwidth difference L1$ and mem: ~20x
- Repeated TLB misses adds a factor ~2-3x
- At least a factor 2-4x is within reach


Copyright 2012 Rogue Wave Software, Inc.

# Cache implementation





Copyright 2012 Rogue Wave Software, Inc.

# Optimizing for cache performance

- Keep the active footprint small
- Use the entire cache line once it has been brought into the cache
- Fetch a cache line prior to its usage
- Let the core that already has the data in its cache do the job
- …


Copyright 2012 Rogue Wave Software, Inc.

# HW prefetching

…a little green man that anticipates your next memory access and prefetches the data to the cache

- Sequential prefetching: Sequential streams [to a page]. Some number of prefetch streams supported. Often only for L2 and L3.
- PC-based prefetching: Detects strides from the same PC. Often also for L1.
- Adjacent prefetching: On a miss, also bring in the "next" cache line. Often only for L2 and L3.


Copyright 2012 Rogue Wave Software, Inc.

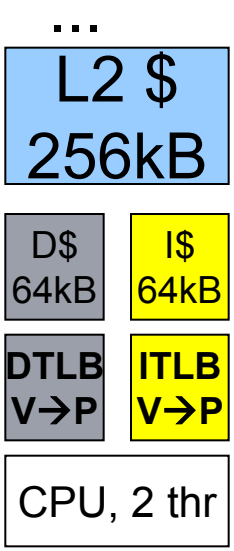
# SW Prefetching

```
/* Unoptimized */
for (j = 0; j < N; j++)
   for (i = 0; i < N; i++)
     x[j][i] = 2 * x[j][i];

/* Optimized */
for (j = 0; j < N; j++)
   for (i = 0; i < N; i++) {
     PREFETCH x[j+1][i]
     x[j][i] = 2 * x[j][i];
   }
```

**(Typically, the HW prefetcher will successfully prefetch sequential streams)**


Copyright 2012 Rogue Wave Software, Inc.

# TLB (address translation cache)




- TLB = cache storing recent virtual →physical translations
- ~1k entries, each translating an entire page
- Cost of a TLB miss is several hundred cycles


Copyright 2012 Rogue Wave Software, Inc.

# Replacement Policies

- LRU:
  Evicts the cache line that was least recently used

- Random replacement:
  A random cache line is selected for eviction


Copyright 2012 Rogue Wave Software, Inc.

# Cache Lingo

- Cacheline: **Data chunk move to/from a cache**
- Associativity: **Number of alternative storage places**
- Replacement policy: **picking the victim (LRU/random)**
- Temporal locality: **Likelihood to access the same data again soon**
- Spatial locality: **Likelihood to access nearby data again soon**

***Typical access pattern:***

**(inner loop stepping through an array)**

**A, B, C, A+4, B, C, A+8, B, C, …**

Copyright 2012 Rogue Wave Software, Inc.

# Cache Lingo

- Miss ratio: **What is the likelihood that a memory access will miss in a cache?**
- Miss rate: **D:o per time unit, e.g. per-second, per-1000-instructions**
- Fetch ratio/rate*): **What is the likelihood that a memory access will cause a fetch to the cache [including HW prefetching]**

- Fetch utilization*): **What fraction of a cacheline was used before it got evicted**
- Writeback utilization*): **What fraction of a cacheline written back to memory contains dirty data**
- Communication utilization*): **What fraction of a communicated cacheline is ever used?**

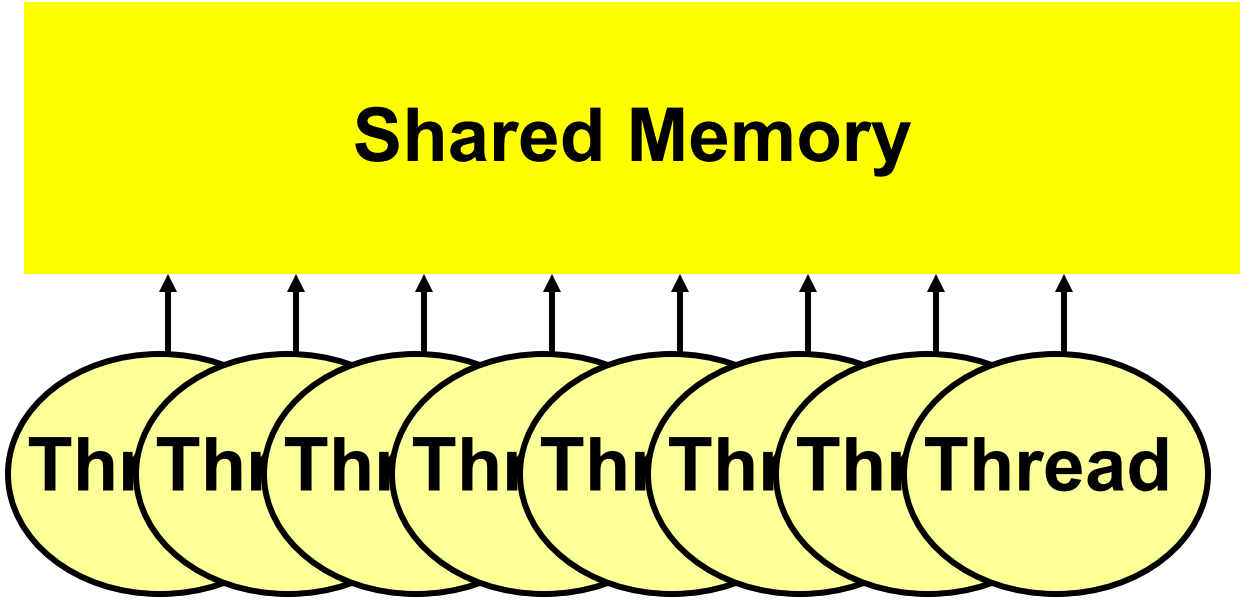
***) This is "Rogue Wave-ish" language**


Copyright 2012 Rogue Wave Software, Inc.

# Cache Coherence


Copyright 2012 Rogue Wave Software, Inc.

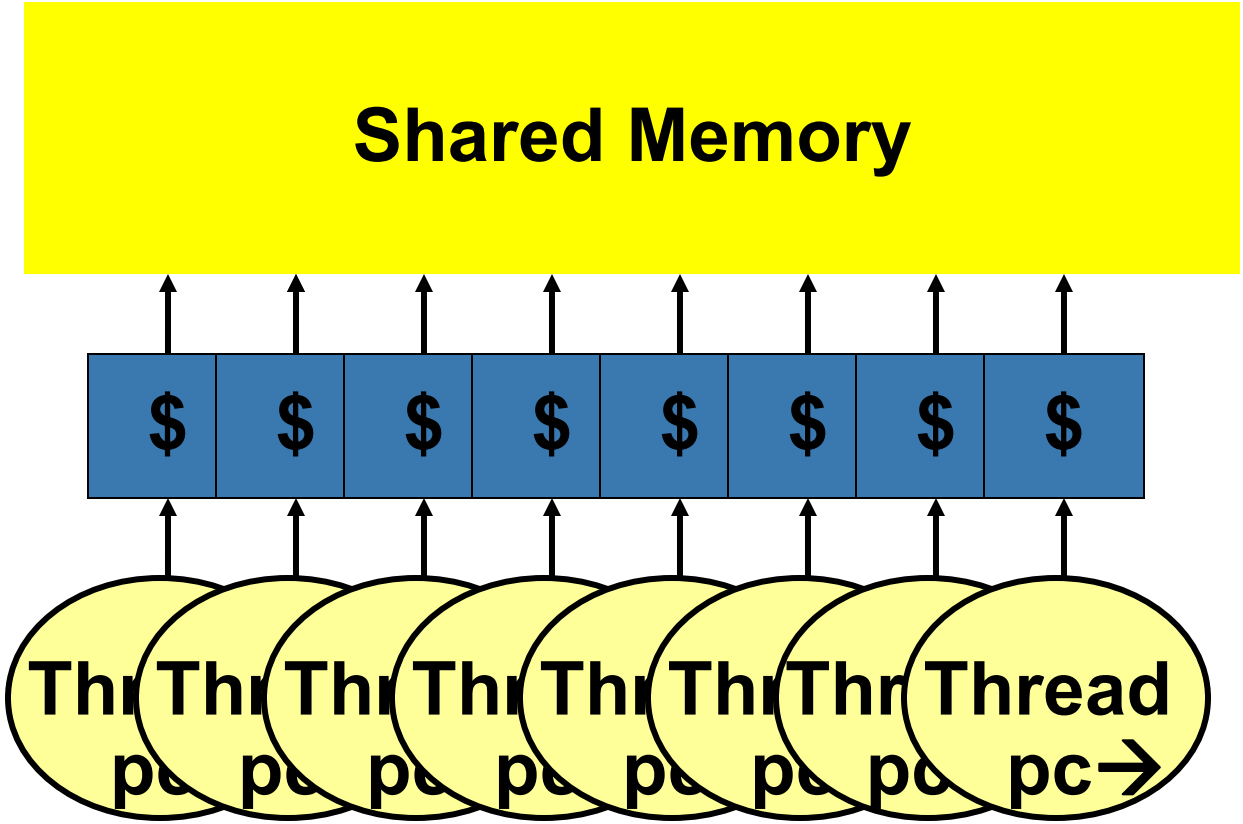
## Programming Model:





Copyright 2012 Rogue Wave Software, Inc.

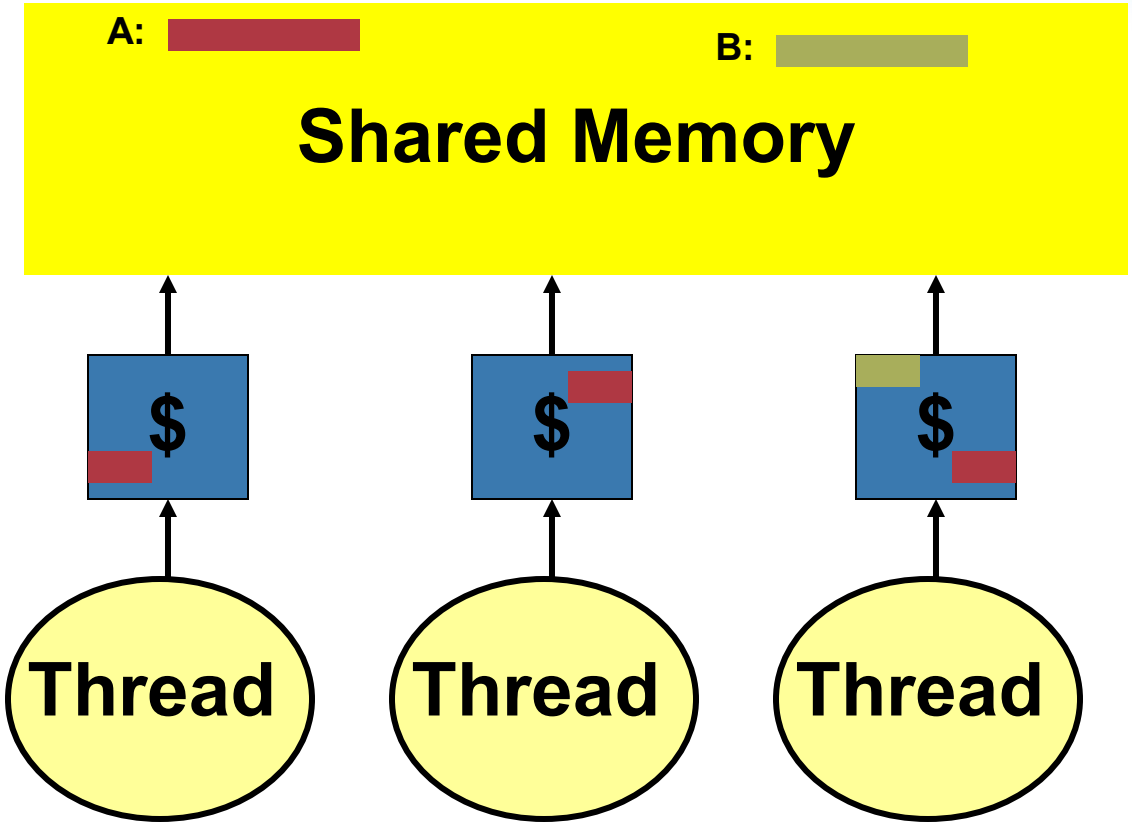
# Adding Caches: More Concurrency





Copyright 2012 Rogue Wave Software, Inc.

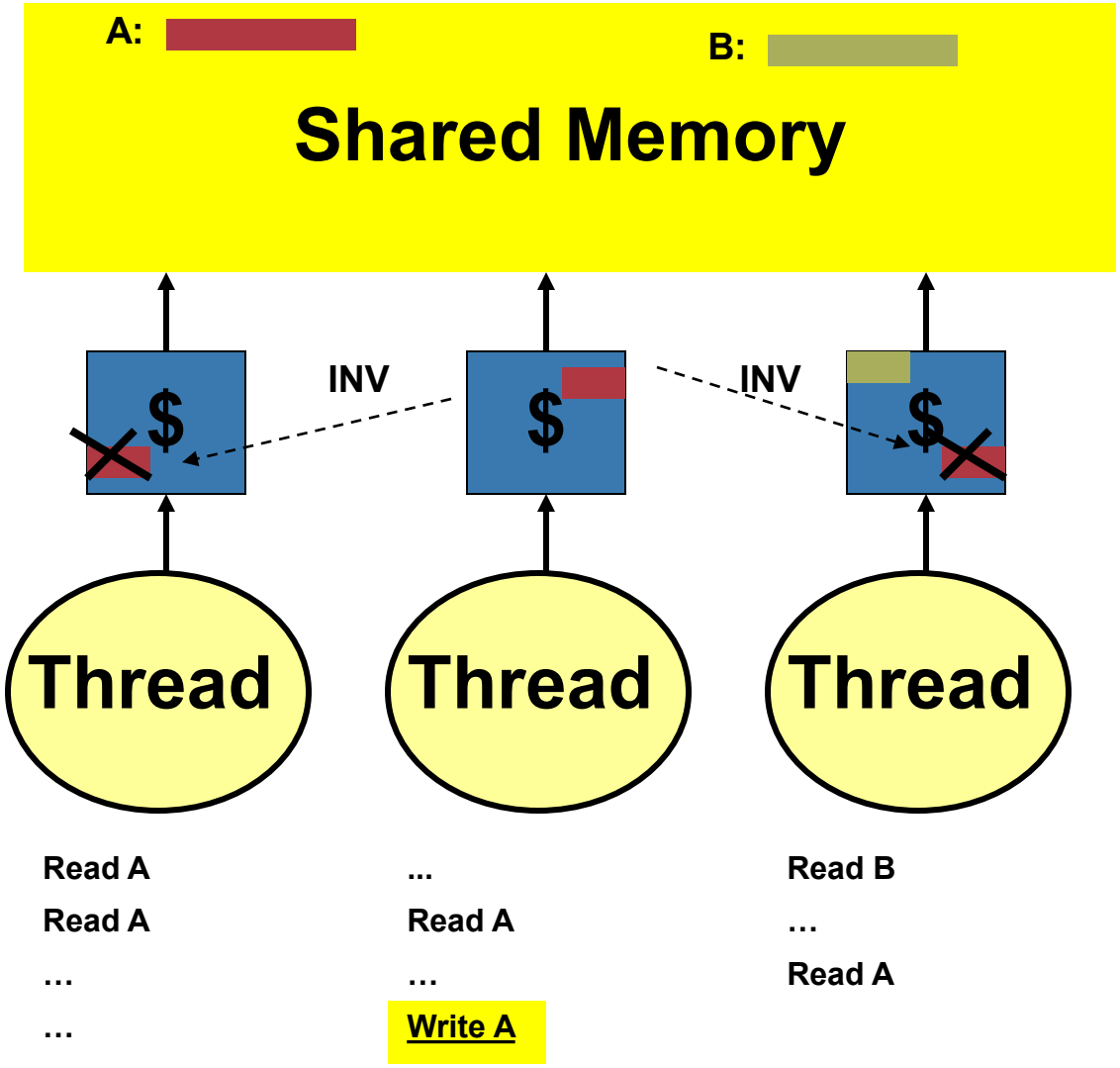
# Caches:
# Automatic Replication of Data




| | | |
|---|---|---|
| Read A | … | Read B |
| Read A | Read A | … |
| … | … | Read A |
| … | | |
| Read A | | |


Copyright 2012 Rogue Wave Software, Inc.

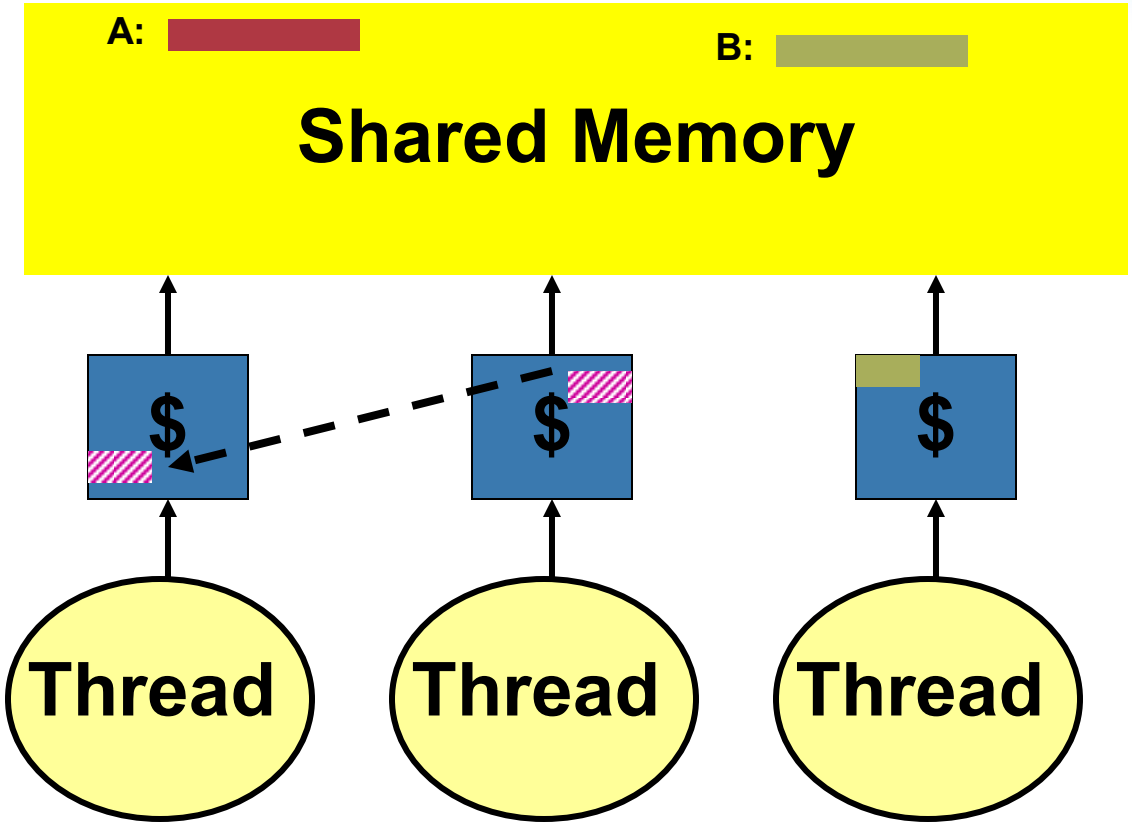
# The Cache Coherent Memory System





Copyright 2012 Rogue Wave Software, Inc.

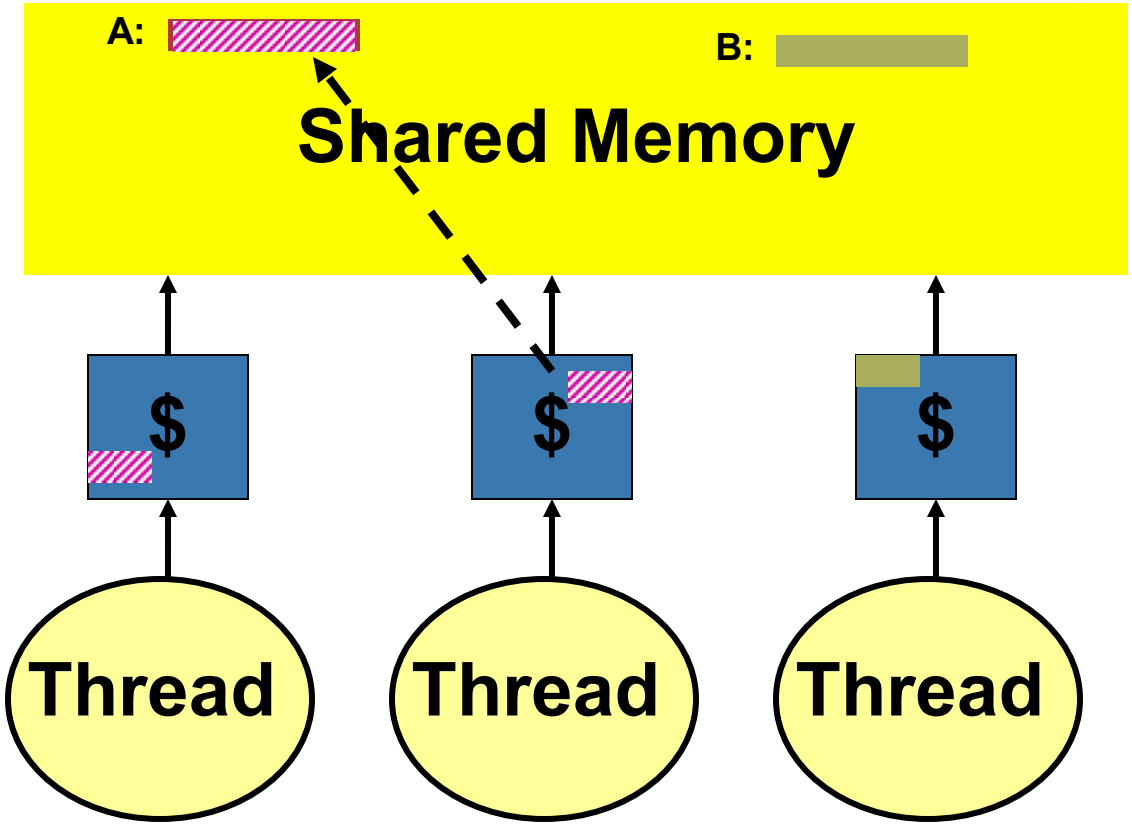
# The Cache Coherent $2$




| | | |
|---|---|---|
| Read A | … | Read B |
| Read A | Read A | … |
| … | … | Read A |
| … | Write A | |
| Read A | | |


Copyright 2012 Rogue Wave Software, Inc.

# Writeback




| | | |
|---|---|---|
| Read A | … | Read B |
| Read A | Read A | … |
| … | … | Read A |
| … | Write A … | |
| Read A | A gets replaced | |


Copyright 2012 Rogue Wave Software, Inc.

# Cache Line States in the MESI Protocol

**M – Modified:**
The data in the cache line is modified and is guaranteed to only reside in this cache. The copy in main memory is not up to date, so when the cache line leaves the modified state the data must be written back to main memory.

**E – Exclusive:**
The data in the cache line is unmodified, but is guaranteed to only reside in this cache.

**S – Shared:**
The data in the cache line is unmodified, and there may also be copies of it in other caches.

**I – Invalid:**
The cache line does not contain valid data.


Copyright 2012 Rogue Wave Software, Inc.

# Summing up Coherence

***Sloppy: There can be many copies of a datum, but only one value***

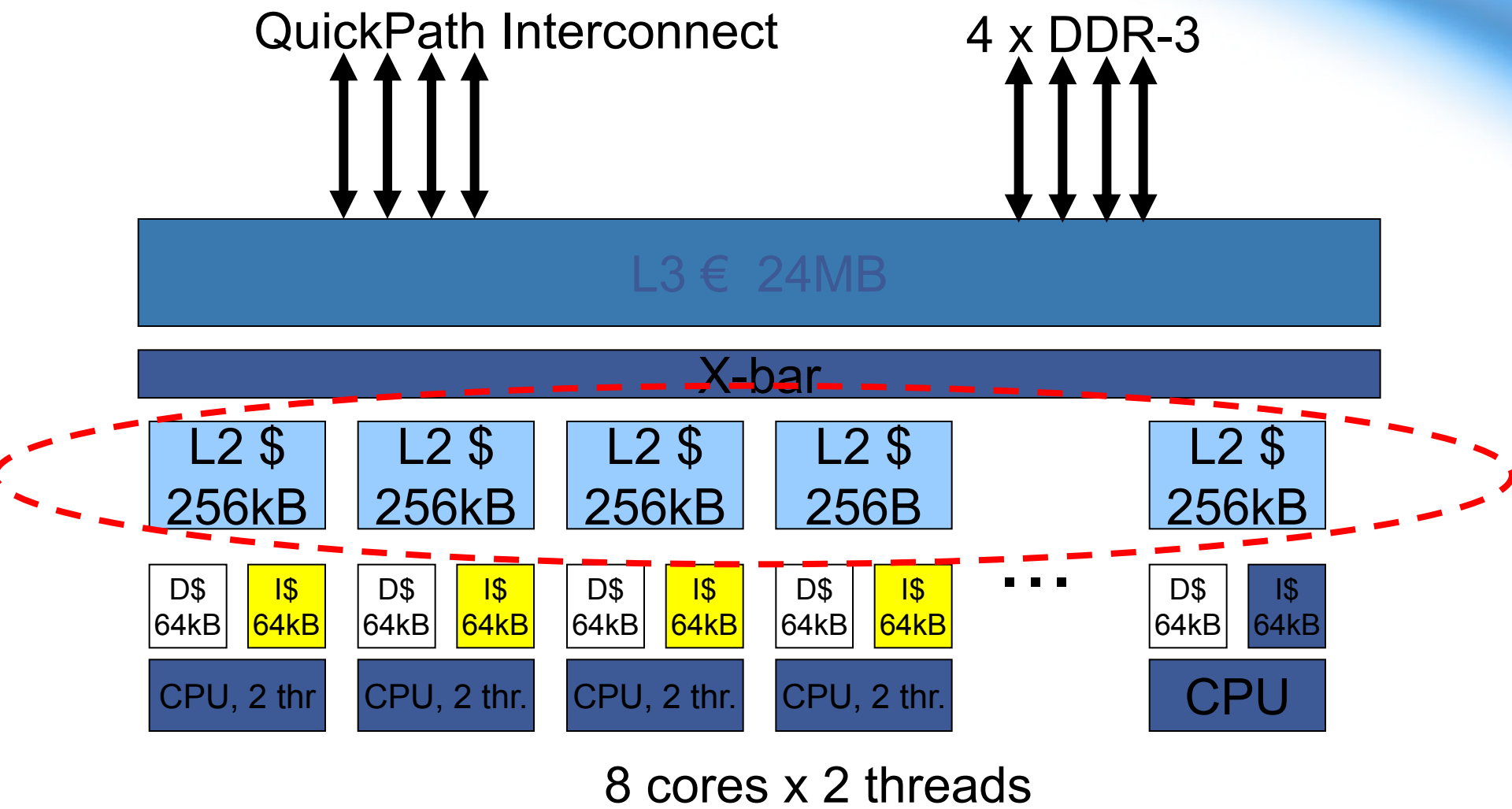
**Too strong definition!**

***Coherence:*** *There is a single global order of value changes to each datum*

***Memory order/model:*** *Defines the order between accesses to many data*


Copyright 2012 Rogue Wave Software, Inc.

# Where does coherence matter?




8 cores x 2 threads


Copyright 2012 Rogue Wave Software, Inc.

# What can go wrong?


Copyright 2012 Rogue Wave Software, Inc.

# Fetch Utilization

- Related to memory layout, spatial locality
  - Using just a subset of fields, array-of-structs (VERY COMMON)
  - Internal alignment problems
  - External alignment problems
  - Dynamic memory allocation
- Related to the algorithm, temporal locality
  - Linked lists, hash tables, maps
- Inappropriate access orders
  - Inefficient Loop Nesting
  - Random access


Copyright 2012 Rogue Wave Software, Inc.

# Partially Used Structure

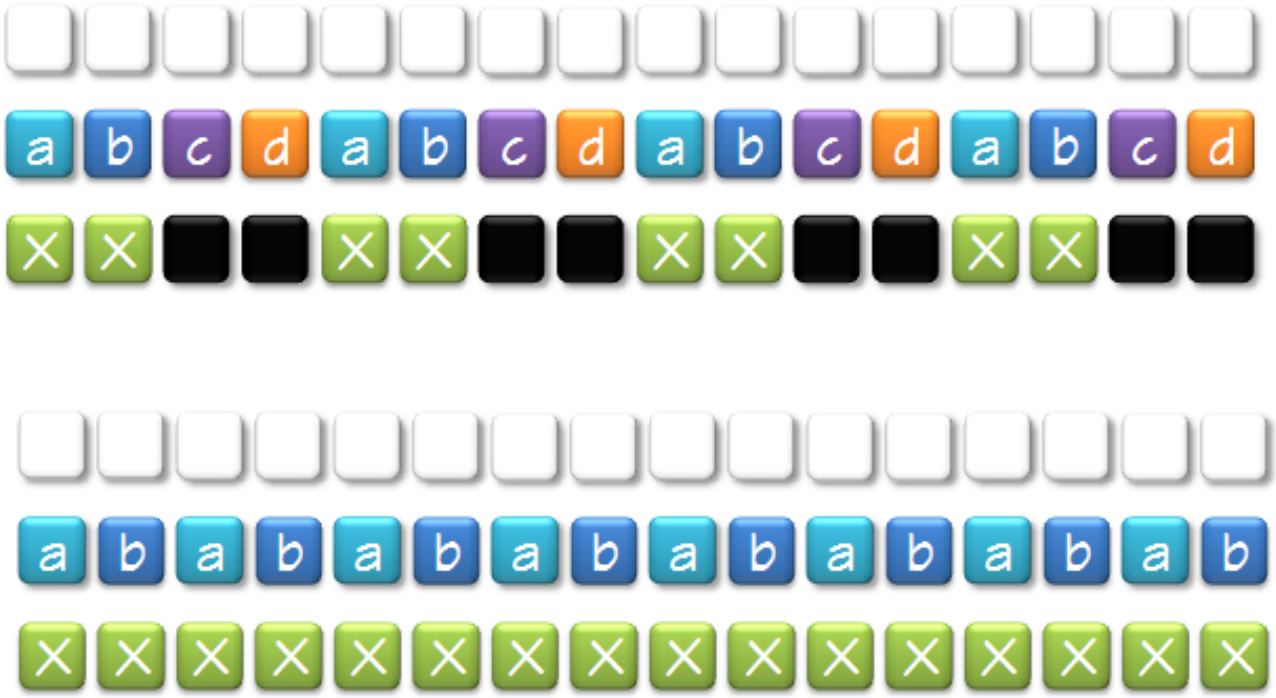
```
struct DATA
{
    int a;
    int b;
    int c;
    int d;
};

DATA * pMyData;
pMyData = new DATA[10*1024*1024];

for (long i=0; i<10*1024*1024; i++)
{
    pMyData[i].a = pMyData[i].b;
}
```

```
struct DATA
{
    int a;
    int b;
};



DATA * pMyData;
pMyData = new DATA[10*1024*1024];

for (long i=0; i<10*1024*1024; i++)
{
    pMyData[i].a = pMyData[i].b;
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Test 0: Partially Used Structure

*Explanation*



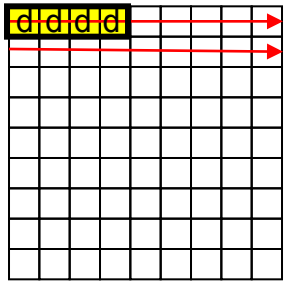


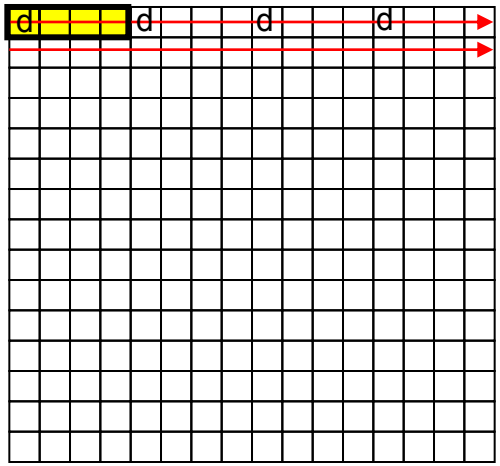
Copyright 2012 Rogue Wave Software, Inc.

# Example: Sparse data

```
//Optimized Example A
for (i=1; i<N; i++) {
  for (j=1; j<N; j++) {
    A_data[i][j]= A_data[i-1][j-1];
  }
}
```

```
//Unoptimized Example A
for (i=1; i<N; i++) {
  for (j=1; j<N; j++) {
    A[i][j].data = A[i-1][j-1].data;
  }
}
```



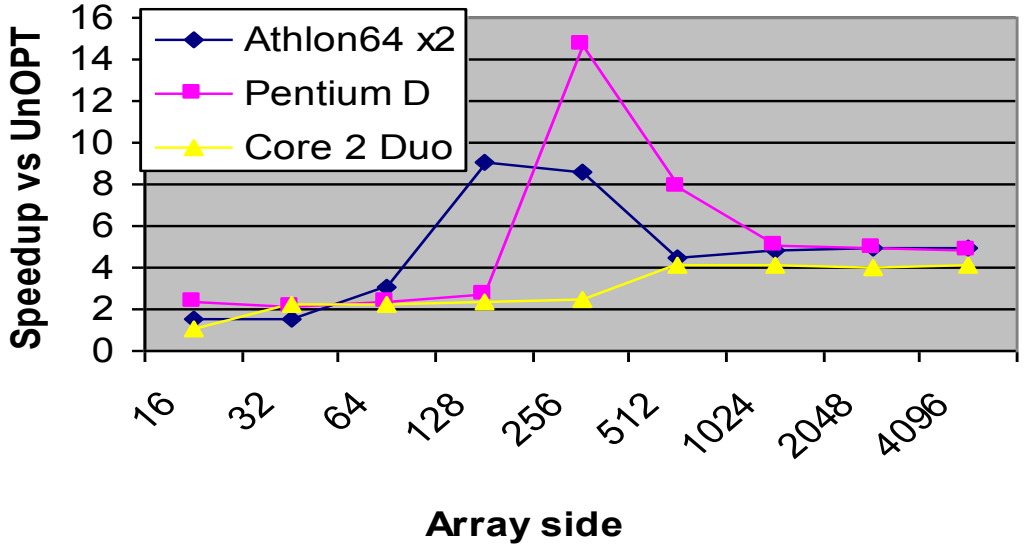





Copyright 2012 Rogue Wave Software, Inc.

# Performance Difference:

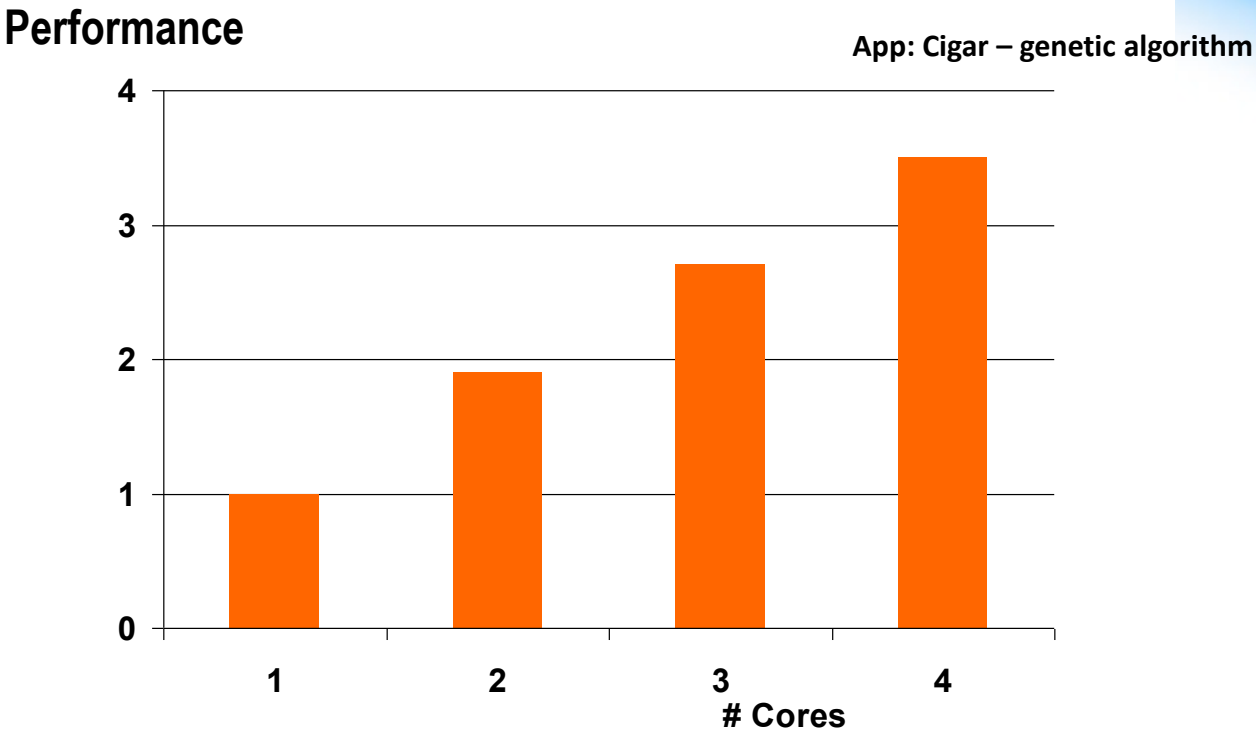
## Sparse Data





Copyright 2012 Rogue Wave Software, Inc.

# Example: A scalable parallel application?

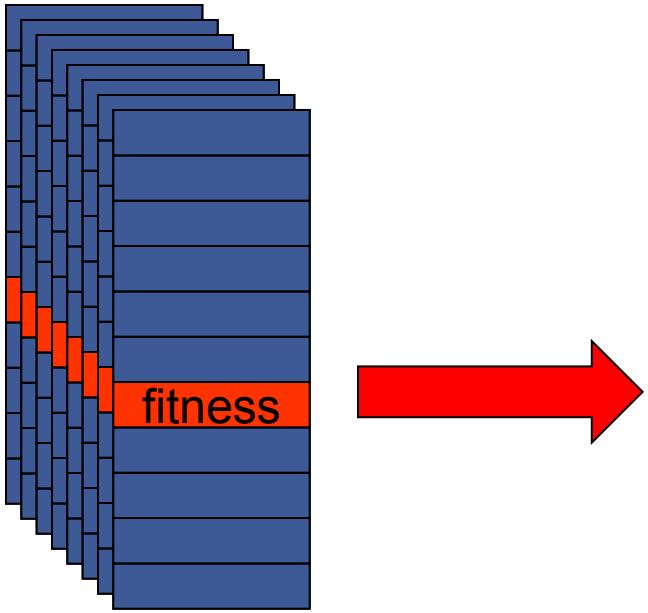



- Looks like a perfect scalable application!
- Are we done?


Copyright 2012 Rogue Wave Software, Inc.

# Cigar: The worst slow spot

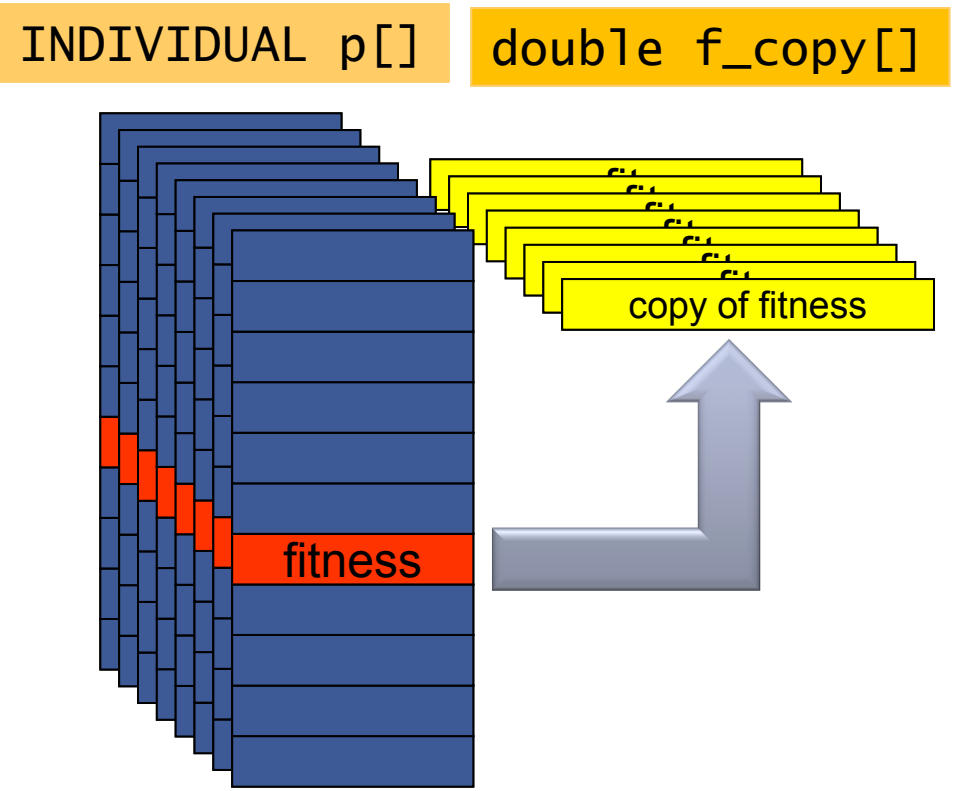
INDIVIDUAL p[]




```
for (current = 0;
    current < howmany-1;
    current++)
{
  max = current;
  /* Find Next best */
  for (i = current+1;
      i < size; i++)
    if((p+rank[i])->fitness>
     (p+rank[max])->fitness)
       max = i;

  SwapInt(&rank[current],
         &rank[max]);
}
```


Copyright 2012 Rogue Wave Software, Inc.

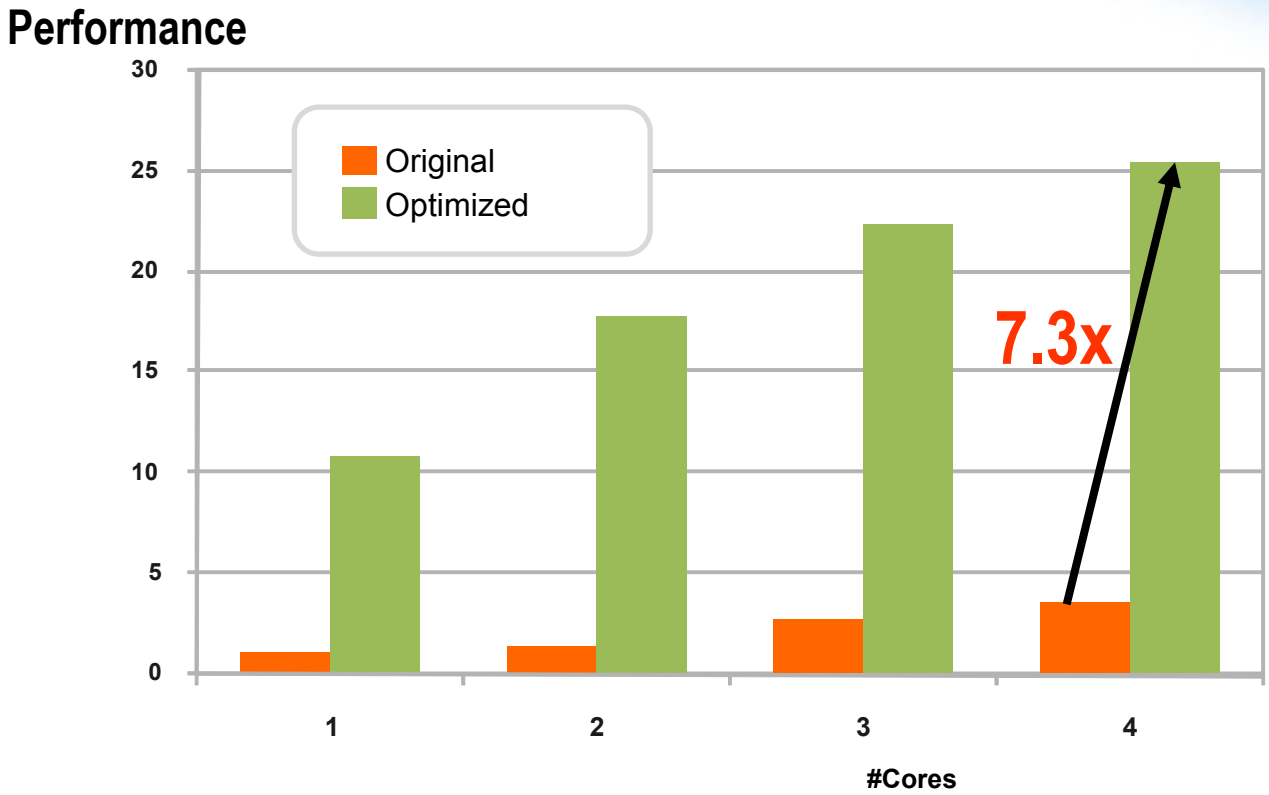
# Cigar: Code change. Duplicate data




```
for (current = 0;
    current < howmany-1;
    current++)
{
  max = current;
  /* Find Next best */
  for (i = current+1;
      i < size; i++)
   if (f_copy[rank[i]] >
       f_copy[rank[max]])
       max = i;

  SwapInt(&rank[current],
         &rank[max]);
}
```


Copyright 2012 Rogue Wave Software, Inc.

# The same application optimized




- **ThreadSpotter advice: Change one data structure**


Copyright 2012 Rogue Wave Software, Inc.

# Alignment Problem

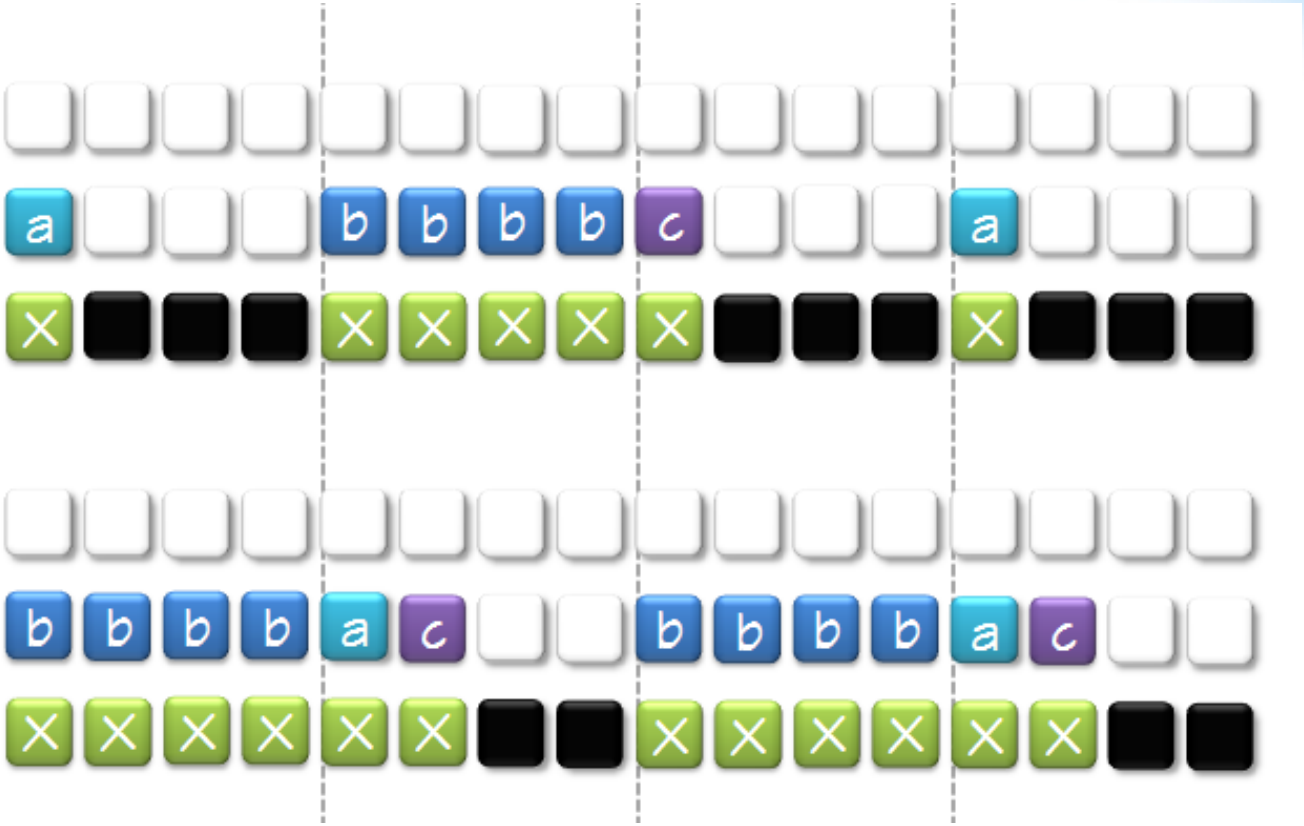
```
struct DATA
{
    char a;
    int b;
    char c;
};

DATA * pMyData;
pMyData = new DATA[36*1024*1024];

for (long i=0; i<36*1024*1024; i++)
{
    pMyData[i].a++;
}
```

```
struct DATA
{
        int b;
        char a;
        char c;
};

DATA * pMyData;
pMyData = new DATA[36*1024*1024];

for (long i=0; i<36*1024*1024; i++)
{
    pMyData[i].a++;
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Test 2: Alignment Problem

*Explanation*





Copyright 2012 Rogue Wave Software, Inc.

# Fetch Utilization – Why is it important?

Three major reasons.

- Reduce the needed cache size to hold the useful data
- Improve the likelihood of finding adjacent data in the cache
- Reduce the memory bus traffic


Copyright 2012 Rogue Wave Software, Inc.

# Fetch Utilization – What to do?

- Split data structures
- Pay attention to alignment rules
- Allocate stuff in larger chunks

- Make temporary copies of just the needed fields.

- Temporal problems requires altering the order you visit data,
- - or rearranging data according to the way your algorithm accesses data.


Copyright 2012 Rogue Wave Software, Inc.

# Spatial Locality – Write-back Utilization

- Only half of 'm' is updated.
- Entire cache lines are still written back.
- 50% write-back utilization.
- Seek a different organization of data that reduces redundant write-backs

```
double m[SIZE][SIZE];
int x, y;

// Half a red-black Gauss-
// Seidel iteration
for (y=1; y<SIZE-1; y++)
   for (x = 1 + y%2;
        x < SIZE-1;
        x += 2)
   m[y][x] = f(m,x,y);
```

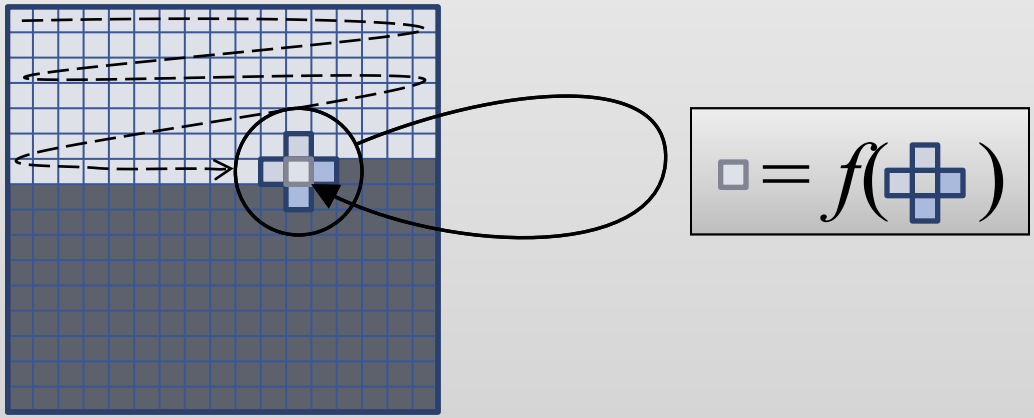

Copyright 2012 Rogue Wave Software, Inc.

# Spatial Locality – Write-back Utilization

The Gauss-Seidel algorithm for solving partial differential equations.

while (! converged) {

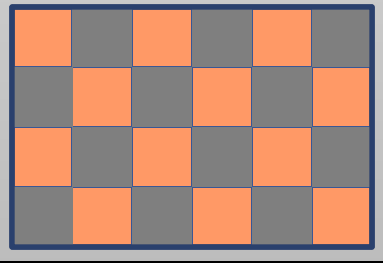
inplace update:



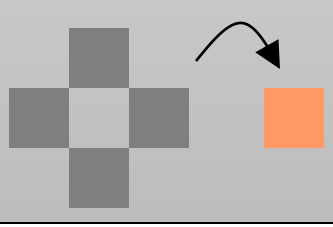

}

Problem: Poor parallelism because of too much data dependency.

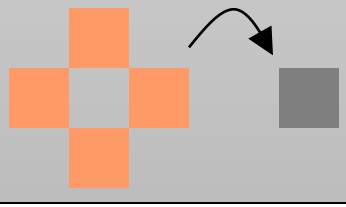
One tweak is the red-black version which updates only every other cell each iteration.



First update all reds:



Then update all blacks:

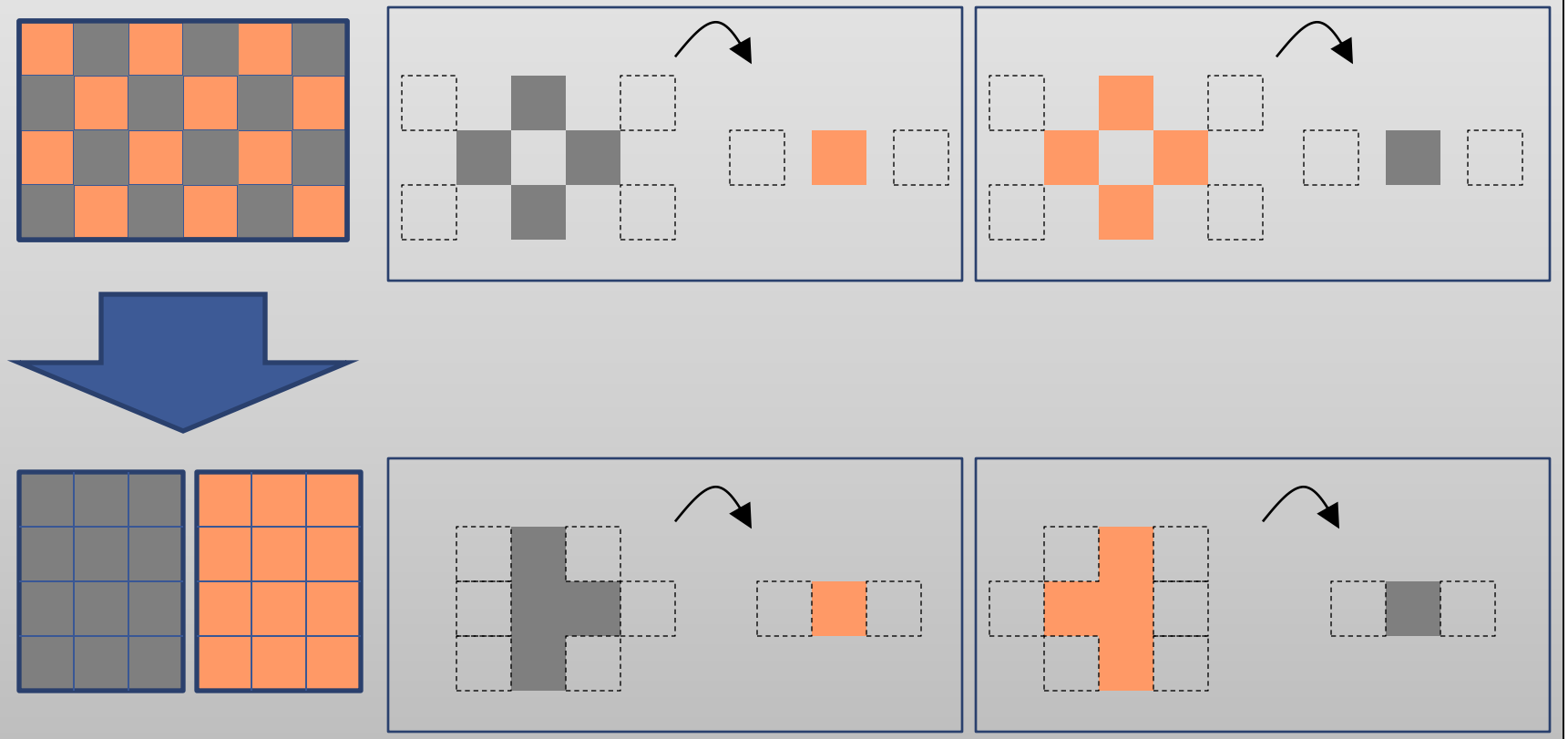



Copyright 2012 Rogue Wave Software, Inc.

# Spatial Locality – Write-back Utilization

The red-black has severe problems in that it visits each cache line twice for each total update. Poor write-back utilization each time.

Idea: Temporarily separate storage of red and black cells, just during solving.




Copyright 2012 Rogue Wave Software, Inc.

# Suggested Fixes

```
double
   m_red[SIZE][SIZE/2],
   m_black[SIZE][SIZE/2];
int x, y;

// Half a red-black Gauss-
// Seidel iteration
for (y=1; y<SIZE-1; y++)
   for (x=1;x<SIZE/2-1;x++)
      m_black[y][x] =
       f(m_red,m_black,x,y);
```

```
double m[SIZE][SIZE];
int x, y;

// Half a red-black Gauss-
// Seidel iteration
for (y=1; y<SIZE-1; y++)
   for (x = 1 + y%2;
        x < SIZE-1;
        x += 2)
   m[y][x] = f(m,x,y);
```


Copyright 2012 Rogue Wave Software, Inc.

# Write-back Utilization – What to do?

- Rearrange data according to the accesses the algorithm is doing
- With good WB-utilization you can go further and use "write-combining." This avoids fetching the cache line in the first place.
  - PREFETCHW


Copyright 2012 Rogue Wave Software, Inc.

# Write-back Utilization - Example

- This transform further opens up for replacing writes with non-temporal writes, as we are not going to need written data for a long time.
- Sample execution on a Core2-Duo T7200:
  - Original 140 s
  - Separated R/B 89 s (improved fetch/wb utilization)
  - With non-temporal stores 71 s (Reduced BW, no fetch needed)


Copyright 2012 Rogue Wave Software, Inc.

# Spatial Locality – Inefficient Loop Nesting

- Warns that a data structure is traversed across cache lines.
- Or some other access pattern that looks similar.

```
double m[SIZE][SIZE];
double colsum[SIZE];

void column_sum() {
   int x, y;
   for (x=0; x<SIZE; x++)
     colsum[x] = 0;
     for (y=0; y<SIZE; y++)
       colsum[x]+= m[y][x];
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Suggested Fixes

```
double m[SIZE][SIZE];
double colsum[SIZE];

void column_sum() {
   int x, y, yy;
   for (x=0; x<SIZE; x++)
     colsum[x] = 0;

   for (yy=0; y<SIZE; y+=B)
    for (x=0; x<SIZE; x++)
     for (y=yy;y<yy+B; y++)
       colsum[x]+= m[y][x];
}
```

```
double m[SIZE][SIZE];
double colsum[SIZE];

void column_sum() {
   int x, y;
   for (x=0; x<SIZE; x++)
     colsum[x] = 0;
     for (y=0; y<SIZE; y++)
       colsum[x]+= m[y][x];
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Inefficient Loop Nesting

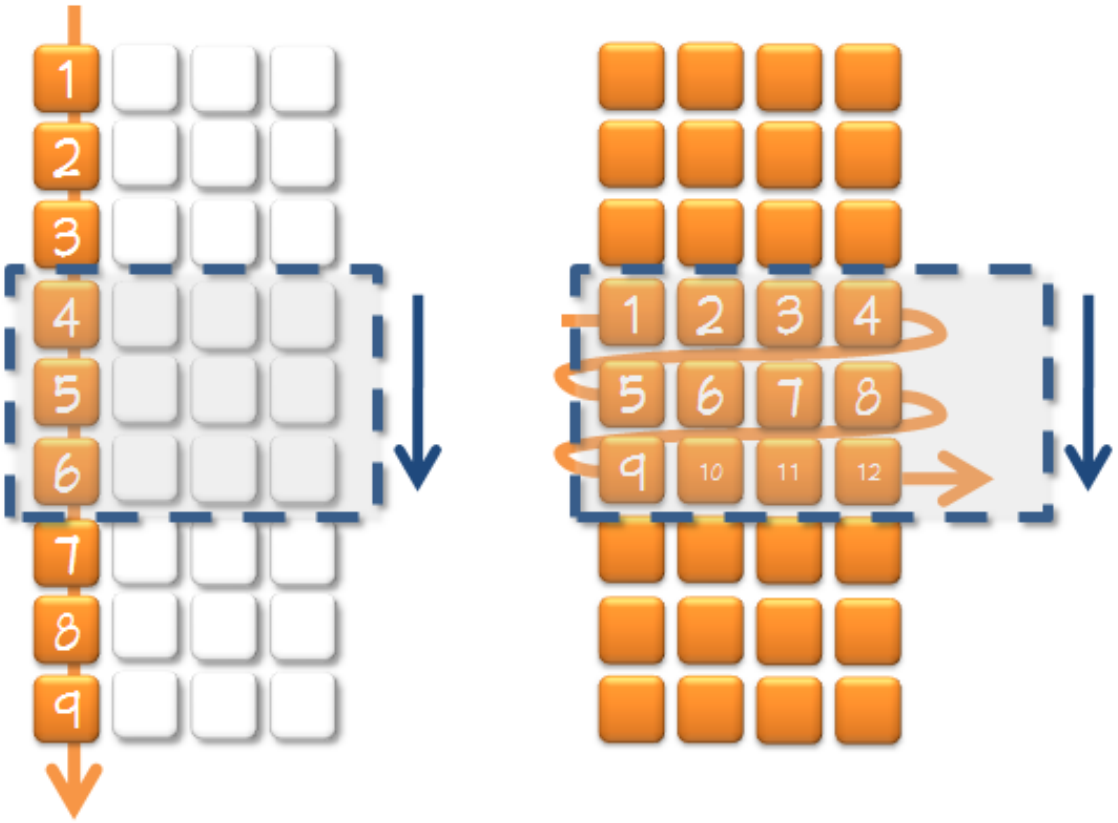
```
#define SIZE (36000*32*32)
#define ROWSIZE 16
#define NBROWS (SIZE/ROWSIZE)

char * p;
p = new char[SIZE];

long nbRows = NBROWS;
long sRowSize = ROWSIZE;

for (long x=0; x<sRowSize; x++)
for (long y=0; y<nbRows; y++)
{
    p[x+y*sRowSize]++;
}
```

```
#define SIZE (36000*32*32)
#define ROWSIZE 16
#define NBROWS (SIZE/ROWSIZE)

char * p;
p = new char[SIZE];

long nbRows = NBROWS;
long sRowSize = ROWSIZE;

for (long y=0; y<nbRows; y++)
for (long x=0; x<sRowSize; x++)
{
    p[x+y*sRowSize]++;
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Test 4: Inefficient Loop Nesting

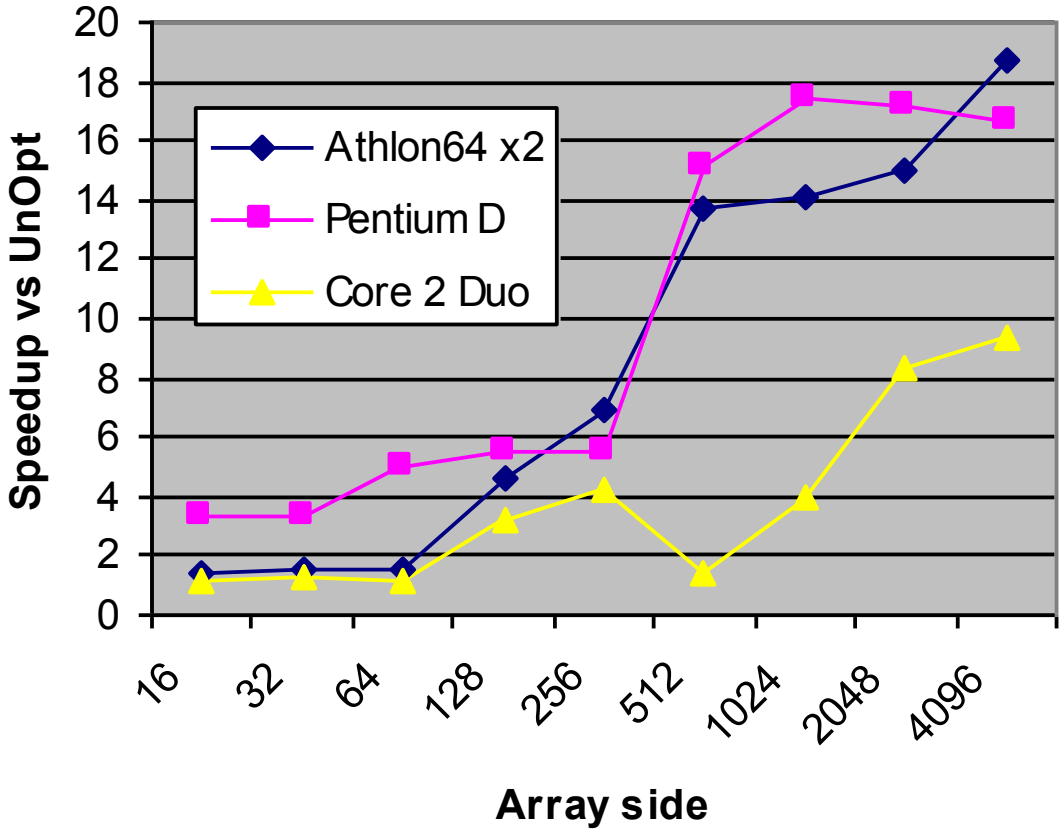
*Explanation*





Copyright 2012 Rogue Wave Software, Inc.

# Performance Difference: Loop order





Copyright 2012 Rogue Wave Software, Inc.

# Inefficient Loop Nesting

- It actually is important even if the cache is large enough
  - Why?
  - The HW prefetcher can only track a few data streams.
- What to do?
  - Change representation
  - Change the code, invert loop nesting. This will affect the access order.
  - Apply blocking. This will also affect the access order.
- Beware of data dependencies.


Copyright 2012 Rogue Wave Software, Inc.

# Random Access

- Both the hash[] lookup, and the linked list traversal jumps around in memory.
- This effectively disables the hardware prefetcher.
- A memory latency hit per access.

```
struct coll_list {
   int key;
   int value;
   coll_list *next;
};
coll_list *hash[SIZE];

int lookup(int key) {
   for(coll_list *p=
       hash[get_hash(key)];
       p; p = p->next)
     if (p->key == key)
       return p->value;
   return -1;
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Suggested Fixes

- Use a dense representation for the collision list, such as a vector, instead of a linked list.
- Keep two parallel vectors, one for keys, one for values.
- Tune the hash table to balance the hits and the inevitable cache miss against the expected collision list size.

```
struct coll_list {
   int key;
   int value;
   coll_list *next;
};
coll_list *hash[SIZE];

int lookup(int key) {
   for(coll_list *p=
       hash[get_hash(key)];
       p; p = p->next)
     if (p->key == key)
       return p->value;
   return -1;
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Random Access – Other cases

- Dynamic memory allocation, dynamic data structures (lists, trees, graphs)
- Dynamic data allocations spreads out data in memory – reducing locality.
- Adds metadata adjacent to memory being allocated – stored next to data, so is transferred, consuming BW and cache space.
- What to do?
  - Linearize. Use one representation while inserting data, then switch to a representation suitable for traversing.
  - Making temporary copies is sometimes acceptable
  - Custom allocators – move metadata out of the way, can help maintaining good allocation order for better locality.


Copyright 2012 Rogue Wave Software, Inc.

# Loop Fusion

- **Fusion advice suggests to merge loop bodies to avoid repeated cache misses.**
- **Loops can be far apart, even in different functions.**

```
struct gadget {
   double length;
   double width;
};
gadget x[SIZE];
double sum_l = 0;
double sum_w = 0;

for (int i=0; i<SIZE; i++)
  sum_l += x[i].length;
// lots of other code ...
for (int i=0; i<SIZE; i++)
  sum_w += x[i].width;
```


Copyright 2012 Rogue Wave Software, Inc.

# Suggested Fixes

```
struct gadget {
   double length;
   double width;
};
gadget x[SIZE];
double sum_l = 0;
double sum_w = 0;

for (int i=0;i<SIZE; i++){
  sum_l += x[i].length;
  sum_w += x[i].width;
}
// lots of other code ...
```

```
struct gadget {
   double length;
   double width;
};
gadget x[SIZE];
double sum_l = 0;
double sum_w = 0;

for (int i=0; i<SIZE; i++)
  sum_l += x[i].length;
// lots of other code ...
for (int i=0; i<SIZE; i++)
  sum_w += x[i].width;
```


Copyright 2012 Rogue Wave Software, Inc.

# Fusion – What to do?

- The objective is to move either the first loop body into the second loop body, or vice versa.
- ThreadSpotter pays attention to data dependencies (R->W, W->R).
- It's the data dependencies between the loops that tell what movement is possible. You can't move a read of a variable to the other side of a write instruction.
- It may also be possible to move things from the second loop into the next iteration of the first loop (or vice versa).


Copyright 2012 Rogue Wave Software, Inc.

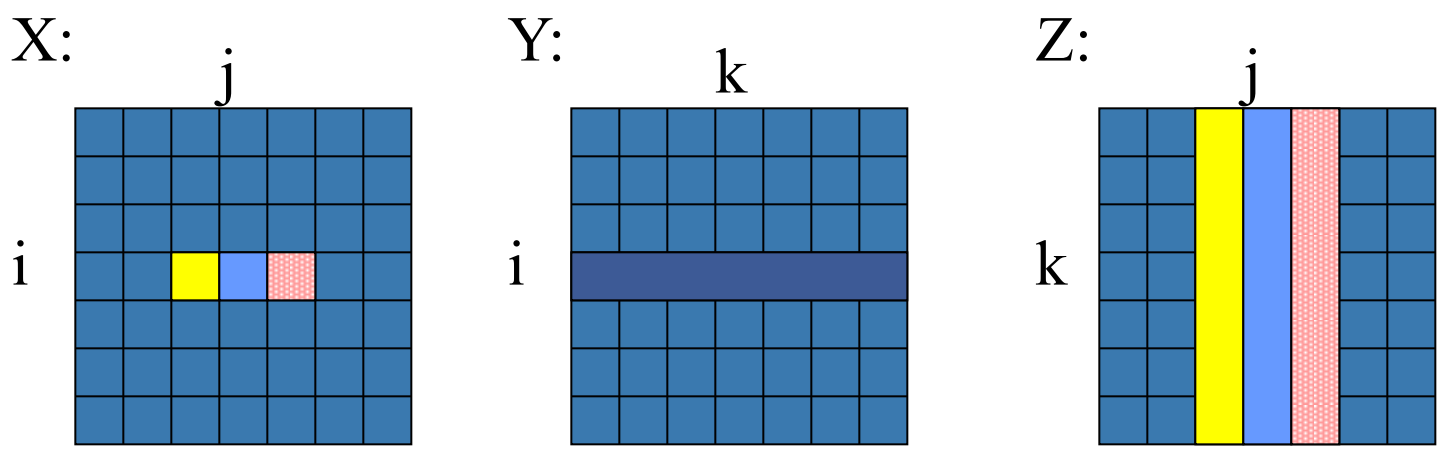
# Matrix Multiplication

```
#define SIZE_MATRIX 256

void multiply(double a[SIZE_MATRIX][SIZE_MATRIX],
              double b[SIZE_MATRIX][SIZE_MATRIX],
              double c[SIZE_MATRIX][SIZE_MATRIX])
{
  for (int i = 0; i < SIZE_MATRIX; i++)
    for (int j = 0; j < SIZE_MATRIX; j++)
      c[i][j] = 0;

  for (int i = 0; i < SIZE_MATRIX; i++)
    for (int j = 0; j < SIZE_MATRIX; j++)
      for (int k = 0; k < SIZE_MATRIX; k++)
        c[i][j] += a[i][k] * b[k][j];
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Blocking

```
/* Unoptimized  ARRAY: x = y * z */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
    {r = 0;
     for (k = 0; k < N; k = k + 1)
        r = r + y[i][k] * z[k][j];
     x[i][j] = r;
    };
```

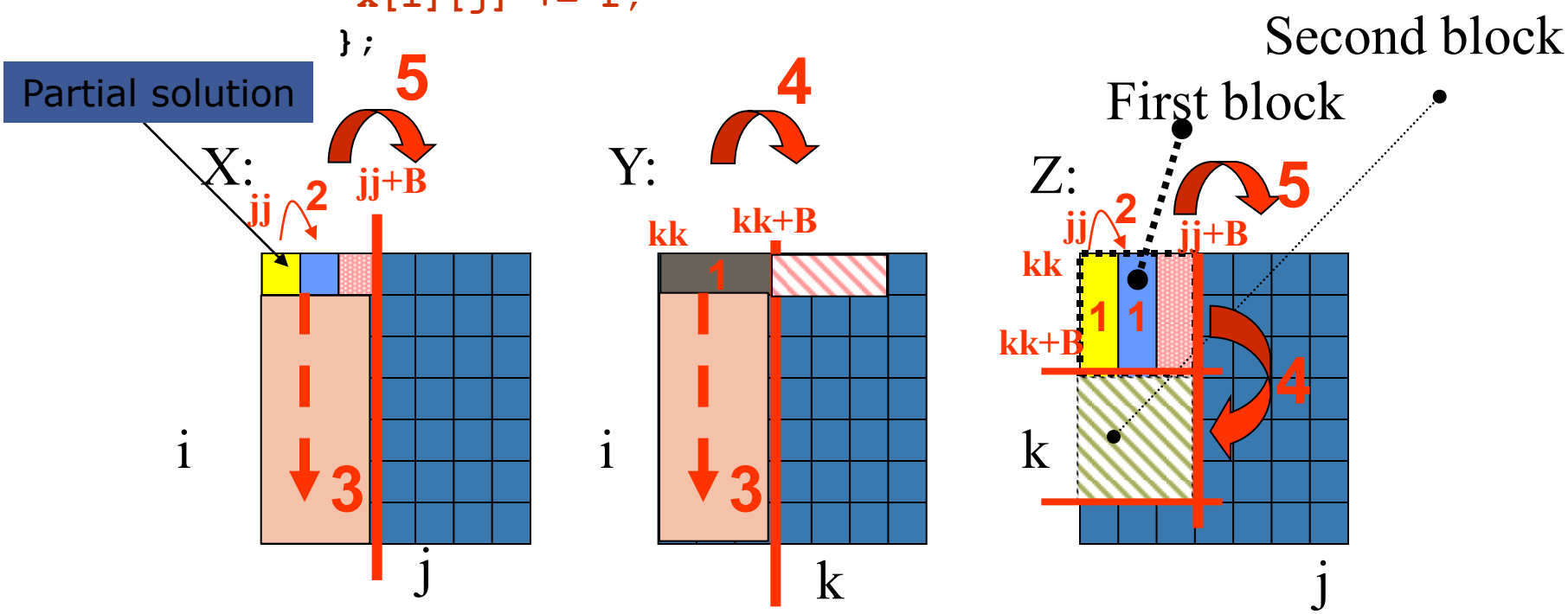




Copyright 2012 Rogue Wave Software, Inc.

# Matrix Multiplication / Blocking

```
/* Optimized  ARRAY: X = Y * Z */
for (jj = 0; jj < N; jj = jj + B)                    /* Loop 5 */
for (kk = 0; kk < N; kk = kk + B)                    /* Loop 4 */
for (i = 0; i < N; i = i + 1)                        /* Loop 3 */
   for (j = jj; j < min(jj+B,N); j = j + 1)          /* Loop 2 */
        {r = 0;
         for (k = kk; k < min(kk+B,N); k = k + 1) /* Loop 1 */
                r = r + y[i][k] * z[k][j];
         x[i][j] += r;
        };
```





Copyright 2012 Rogue Wave Software, Inc.

# Matrix Multiplication / Blocking

```
#define SIZE_MATRIX 256
#define BLOCK_I 16
#define BLOCK_K 16
#define BLOCK_J 16

void multiply(double a[SIZE_MATRIX][SIZE_MATRIX],
              double b[SIZE_MATRIX][SIZE_MATRIX],
              double c[SIZE_MATRIX][SIZE_MATRIX])
{
  for (int i = 0; i < SIZE_MATRIX; i++)
      for (int j = 0; j < SIZE_MATRIX; j++)
          c[i][j] = 0;

  for (int ii = 0; ii < SIZE_MATRIX; ii += BLOCK_I)
    for (int kk = 0; kk < SIZE_MATRIX; kk += BLOCK_K)
      for (int jj = 0; jj < SIZE_MATRIX; jj += BLOCK_J)
        for (int i = ii; i < ii + BLOCK_I && i < SIZE_MATRIX; i++)
          for (int k = kk; k < kk + BLOCK_K && k < SIZE_MATRIX; k++)
            for (int j = jj; j < jj + BLOCK_J && j < SIZE_MATRIX; j++)
              c[i][j] += a[i][k] * b[k][j];
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Blocking – What to do?

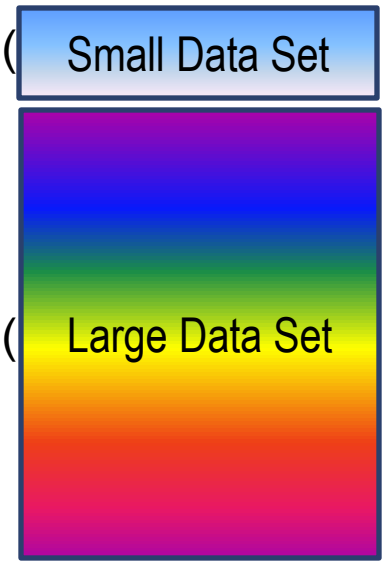
- Blocking is another way of saying: batch up your operations
- Use nerd curves to find the active footprint size. Combine this with the target cache size to find a necessary blocking factor.
- Pay attention to the dimensionality of the data.
- Blocking is going to affect the order of operations.
- ThreadSpotter is trying to avoid giving blocking advice if there are loop-carried data dependencies.

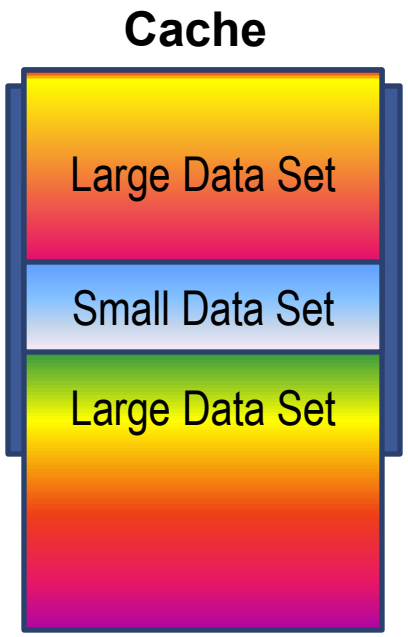

Copyright 2012 Rogue Wave Software, Inc.

# Cache Pollution

- Cache pollution occurs when a data set, larger than the cache, is accessed a lot.
- It will fill up the cache many times over with data that is never going to be reused.

```
for () {
  touch(
  touch(
}
```



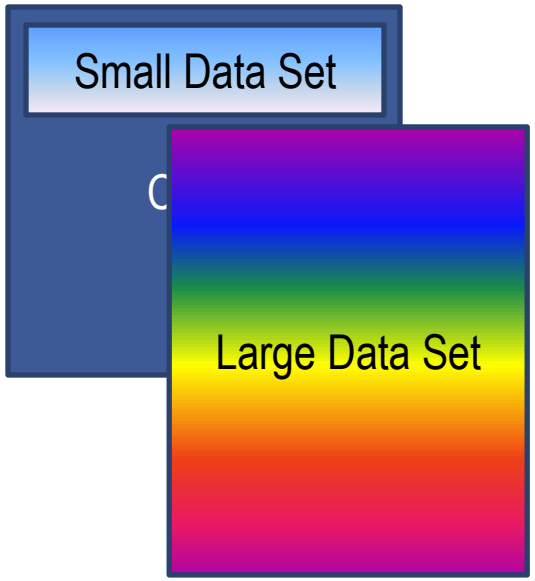


Copyright 2012 Rogue Wave Software, Inc.

# Suggested Fixes

- **Consider using "cache bypass" on the entire "Large Data Set"**
- **As an extra bonus, try using cache bypass only for the part of the large data set that won't fit in the cache**

```
for () {
  touch(Small Data Set);

  touch(Large Data Set, "bypass_cache");

}
```





Copyright 2012 Rogue Wave Software, Inc.

# Cache Pollution – What to do?

- Some architectures have instructions that bypass the cache when fetching or storing data.
- This allows smaller data structures to remain in the cache.
- Non-temporal write (x86, SSE2: MOVNTDQ)
- Non-temporal prefetch (x86: PREFETCHNTA)
- Write prefetch for (x86: PREFETCHW)
- Processor specific.
  For instance, PREFETCHNTA doesn't work on Nehalem.


Copyright 2012 Rogue Wave Software, Inc.

# Temporal Data

```
void temporal_copy(char* outbuff, char* inbuff, int size)  {
    const int step = 64; // cache line
    while (size > step)  {
        _mm_prefetch(inbuff + 320, _MM_HINT_T0); // temporal prefetch
        __m128i A = _mm_loadu_si128((__m128i*) (inbuff +  0));
        __m128i B = _mm_loadu_si128((__m128i*) (inbuff + 16));
        __m128i C = _mm_loadu_si128((__m128i*) (inbuff + 32));
        __m128i D = _mm_loadu_si128((__m128i*) (inbuff + 48));

        _mm_store_si128((__m128i*) (outbuff +  0), A);
        _mm_store_si128((__m128i*) (outbuff + 16), B);
        _mm_store_si128((__m128i*) (outbuff + 32), C);
        _mm_store_si128((__m128i*) (outbuff + 48), D);

        inbuff  += step;
        outbuff += step;
        size -= step;
    }
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Non Temporal Data

```
void nontemporal_copy(char* outbuff, char* inbuff, int size)  {
    const int step = 64; // cache line
    while (size > step)  {
        _mm_prefetch(inbuff + 320, _MM_HINT_NTA); // non-temporal prefetch
        __m128i A = _mm_loadu_si128((__m128i*) (inbuff +  0));
        __m128i B = _mm_loadu_si128((__m128i*) (inbuff + 16));
        __m128i C = _mm_loadu_si128((__m128i*) (inbuff + 32));
        __m128i D = _mm_loadu_si128((__m128i*) (inbuff + 48));

        // destination address must be 16-byte aligned!
        _mm_stream_si128((__m128i*) (outbuff +  0), A);
        _mm_stream_si128((__m128i*) (outbuff + 16), B);
        _mm_stream_si128((__m128i*) (outbuff + 32), C);
        _mm_stream_si128((__m128i*) (outbuff + 48), D);

        inbuff  += step;
        outbuff += step;
        size -= step;
    }
    _mm_mfence();
}
```


Copyright 2012 Rogue Wave Software, Inc.

# False Sharing

- Several threads update logically different data, accidentally residing in the same physical cache line.
- Cache line will repeatedly be communicated between caches.

```
int *partial_sum = new int[
   omp_get_num_threads()];
int i;
#omp parallel for
   private(i, x)
for (int y=0;y<SIZE;y++) {
   i=omp_get_thread_num();
   for (int x=0;x<SIZE;x++)
     partial_sum[i] +=
       f(x, y);
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Suggested Fixes

```
int sum;
#omp parallel for
   reduction(+: sum)
   private(x)
for (int y=0;y<SIZE;y++) {
   for (int x=0;x<SIZE;x++)
      sum += f(x, y);
}
```

- or make partial_sum more sparse, at most one element per cache line by adding dummy elements.

```
int *partial_sum = new int[
   omp_get_num_threads()];
int i;
#omp parallel for
   private(i, x)
for (int y=0;y<SIZE;y++) {
   i=omp_get_thread_num();
   for (int x=0;x<SIZE;x++)
      partial_sum[i] +=
         f(x, y);
}
```


Copyright 2012 Rogue Wave Software, Inc.

# False Sharing

```
int sum1;
int sum2;

void thread1(int v[], int v_count) {
sum1 = 0;
for (int i = 0; i < v_count; i++)
sum1 += v[i];
}

void thread2(int v[], int v_count) {
sum2 = 0;
for (int i = 0; i < v_count; i++)
sum2 += v[i];
}
```


Copyright 2012 Rogue Wave Software, Inc.

# False Sharing (Fixed)

```
int __attribute__((aligned(64))) sum1;
int __attribute__((aligned(64))) sum2;

void thread1(int v[], int v_count) {
sum1 = 0;
for (int i = 0; i < v_count; i++)
  sum1 += v[i];
}

void thread2(int v[], int v_count) {
sum2 = 0;
for (int i = 0; i < v_count; i++)
  sum2 += v[i];
}
```


Copyright 2012 Rogue Wave Software, Inc.

# False Sharing

- Keep items used in separate threads in their own cache line chunk of memory.
- Beware of vectors & arrays for shared data, such as locks (spinlocks, mutexes, etc.)
- Pad data when necessary.


Copyright 2012 Rogue Wave Software, Inc.

# Communication Utilization

- When passing data between caches (threads), make sure each cache line is fully written/used.
- Otherwise, there will be unnecessarily many roundtrips.

```
// thread 1 - producer
while (1) {
   int x = ...;
   queue.push(x);
   not_empty.signal();
}

// thread 2 – consumer
while (1) {
   not_empty.wait();
   int x = queue.pop();
   // do stuff with x
}
```


Copyright 2012 Rogue Wave Software, Inc.

# Suggested Fixes

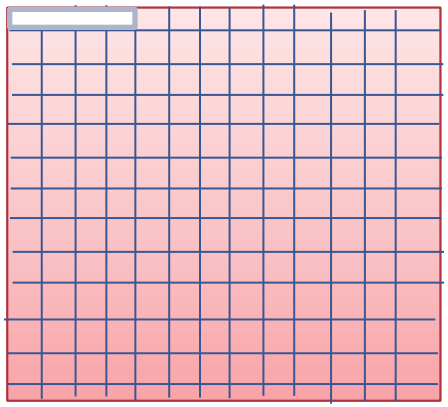
```
// thread 1 - producer
while (1) {
   for (int i=0; i<16;i++){
     int x = ...;
     queue.push(x);
   }
   not_empty.signal();
}

// thread 2 – consumer
while (1) {
   not_empty.wait();
   while (queue.empty()) {
     int x = queue.pop();
     // do stuff with x
} }
```

```
// thread 1 - producer
while (1) {
   int x = ...;
   queue.push(x);
   not_empty.signal();
}

// thread 2 – consumer
while (1) {
   not_empty.wait();
   int x = queue.pop();
   // do stuff with x
}
```

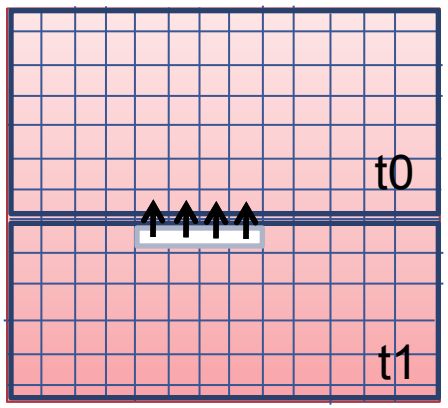

Copyright 2012 Rogue Wave Software, Inc.

# Communication Utilization – What to do?

- Let the producer(s) fill up cache lines before signaling the consumer.
- Pay attention to cache line borders when partitioning data



data set, cache line

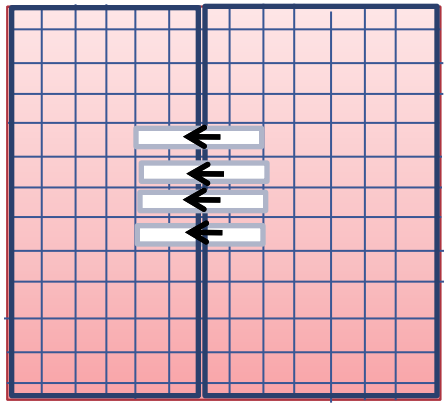


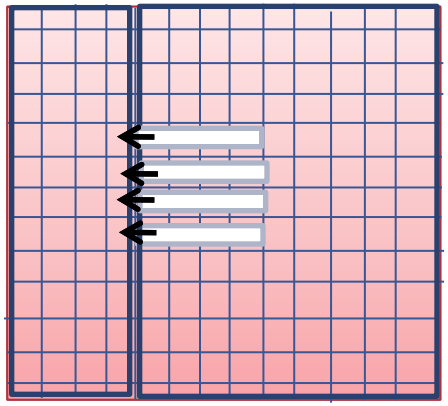

several threads
Good behavior


Copyright 2012 Rogue Wave Software, Inc.

# Communication Utilization – poor examples

- Poor examples include those where just part of a cache line gets used in the other thread.
- Make sure cache lines don't straddle a domain boundary.





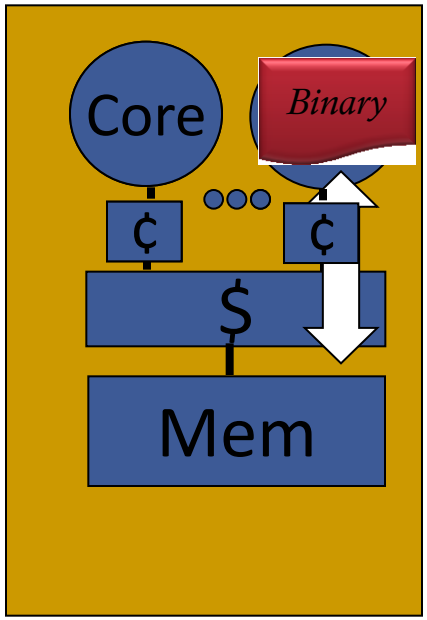
straddling cache line
False sharing,
poor communication utilization


Copyright 2012 Rogue Wave Software, Inc.

# Workflow


Copyright 2012 Rogue Wave Software, Inc.

# 1. Optimize for Cache/Memory




1st Order MC Performance Problems

- Limited cache capacity
- Deep cache hierarchy
- Slow DRAM latency
- Low DRAM bandwidth

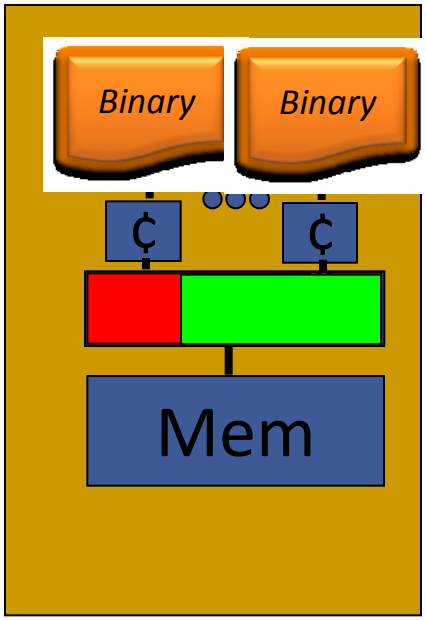
## Issues to deal with:

- Data allocation schema
- Temporal data re-use
- Spatial data usage
- Cache utilization
- HW/SW prefetching issues

…


Copyright 2012 Rogue Wave Software, Inc.

# 2. Remove Cache Waste




1st Order MC Performance Problems

- Additional multicore issues:
  - Even less cache resources per application
  - Sharing of cache resources
  - Wasted cache usage

## Issues to deal with

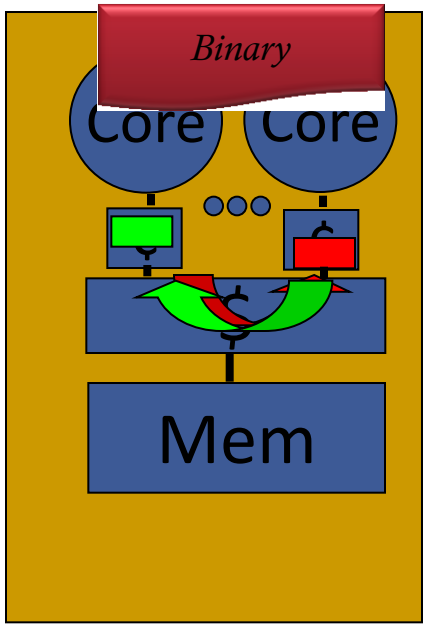
All previous issues and:

- Cache sharing effects
- Cache pollution side-effects
- Optimal usage of cache resources

…


Copyright 2012 Rogue Wave Software, Inc.

# 3. Optimize Parallel Code

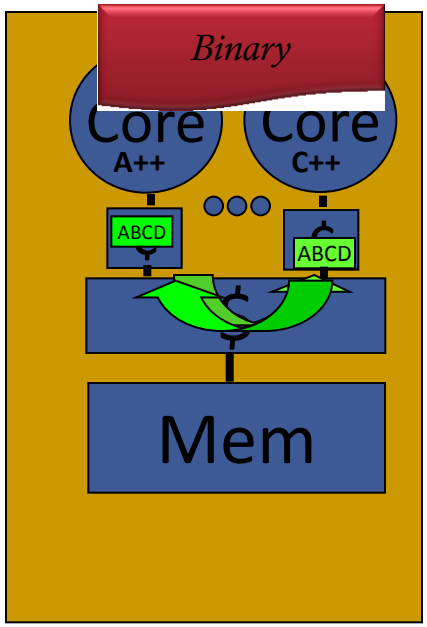



1st Order MC Performance Problems

- Thread interaction
  - Coherence traffic
  - Producer/consumer sharing
  - False sharing ...


Copyright 2012 Rogue Wave Software, Inc.

# 3. Optimize Parallel Code




1st Order MC Performance Problems

- Thread interaction
  - Coherence traffic
  - Producer/consumer sharing
  - False sharing ...

## Issues to deal with

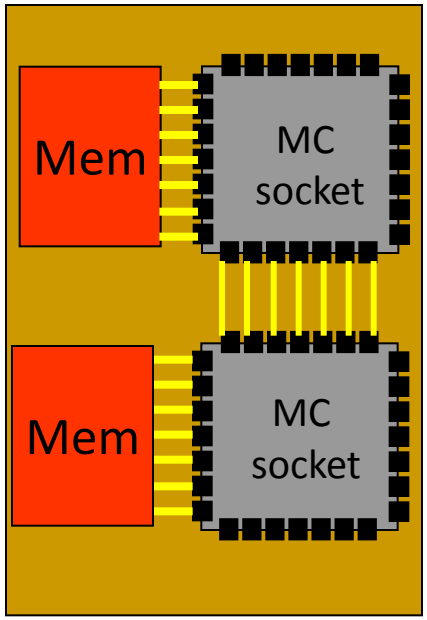
All previous issues and:

- Allocation/division of shared data
- Inter-thread data dependence
- Inter-thread cache sharing
- Communication patterns

...


Copyright 2012 Rogue Wave Software, Inc.

# 4. Place Threads/Memory




1st Order MC Performance Problems

- Core2 Quad, Magny Cours, Multisocket
- Non-uniform memory access (NUMA)
- Non-uniform communication cost (NUCA)
- Higher costs everywhere…

## Issues to deal with:

All previous issues become more important, plus

- Job scheduling issues
- Thread to core binding issues
- Memory placement issues


Copyright 2012 Rogue Wave Software, Inc.

# Rogue Wave ThreadSpotter™

Source:

C, C++, Fortran, OpenMP…

```
/* Unoptimized  Array Multiplication:  x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k]  * z[k][j];
      x[i][j] = r;
      }
/* Unoptimized  Array Multiplication:  x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k]  * z[k][j];
      x[i][j] = r;
      }
```

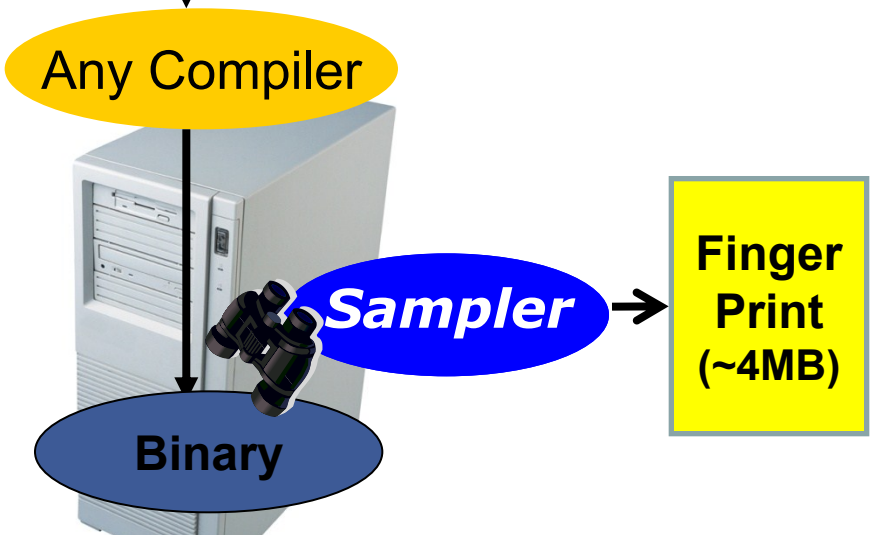
**Mission:**

**Find the SlowSpots™**

**Asses their importance**

**Enable for non-experts to fix them**

**Improve the productivity of performance experts**




**Host System**


Copyright 2012 Rogue Wave Software, Inc.

# Rogue Wav

Source:

C, C++, F

```
/* Unoptimized  Array Multiplication: x = y
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k]  * z[k][j];
      x[i][j] = r;
      }
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k]  * z[k][j];
      x[i][j] = r;
      }
```

What?

Where?

How?

Help!

Any Compiler

Binary

Sampler → Finger Print (~4MB) → Analysis → Advice

Target System Parameters

Host System


Copyright 2012 Rogue Wave Software, Inc.

# Loop Focus Tab

Acumem VPE: ./art (64k/64) - Mozilla Firefox

Arkiv Redigera Visa Historik Bokmärken Verktyg

file:///C:/Document

Summary | Loops | Bandwidth Issues | Latency Issues | Files | Execution | About/Hel

| Loop | % of misses | % of fetches | Utilization | Issues |
|---|---|---|---|---|
| 1 | 85.8% | 62.3% | 17.7% | |
| 2 | 9.5% | 7.7% | 23.8% | |
| 4 | 0.0% | 6.1% | 34.1% | |
| 3 | 0.0% | 4.4% | 23.7% | |
| 6 | 0.0% | 4.2% | 36.8% | |
| 7 | 0.0% | 4.2% | 25.1% | |
| 5 | 0.0% | 4.0% | 26.1% | |
| 8 | 4.2% | 3.2% | 18.4% | |
| 9 | 0.0% | 1.1% | 23.5% | |

Cache line utilization
Inefficient loop nesting
Random access pattern

List of bad loops

Loop 1

- Loop statistics
- Loop instructions
- Instruction groups in this loop, summary of issues
- Instruction group 1
- Instruction group 2
- Instruction group 3

Copyright (c) 2007 Acumem AB. All Rights Reserved. Patents pending.

Explaining what to do

```
600          tnorm += f1_layer[ti].P * f1_layer[ti].P;
601
602          if (ttemp != f1_layer[ti].P)
603             tresult=0;
604       }
605       f1res = tresult;
606
607       /* Compute F1 - Q values */
608
609       tnorm = sqrt((double) tnorm);
610       for (tj=0;tj<numf1s;tj++)
611 4.4%        f1_layer[tj].Q = f1_layer[tj].P;
612
613       /* Compute F2 - y values */
614       for (tj=0;tj<numf2s;tj++
615       {
616         Y[tj].y = 0;
617         if ( !Y[tj].reset )
618         for (ti=0;ti<numf1s;ti++)
619 65.4%        Y[tj].y += f1_layer[ti].P * bus[ti][tj];
620       }
621
622       /* Find match */
623       winner = 0;
624       for (ti=0;ti<numf2s;ti++)
625       {
626        if (Y[ti].y > Y[winner].y)
627           winner =ti;
628       }
629
630
631    }
632 #ifdef DEBUG
633    if (DB1) print_f12();
634    if (DB1) printf("\n num iterations for p to stabilize = %i \n",j);
635 #endif
636    match_confidence=simtest2();
637    if ((match_confidence) > rho)
638    {
```

Spotting the crime

Klar

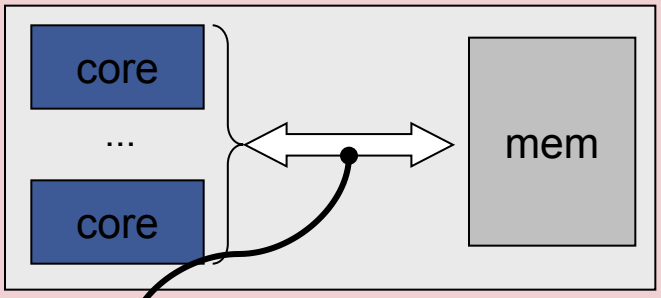

Copyright 2012 Rogue Wave Software, Inc.

# Insight and Efficiency

Slowdown 10% (for long-running applications)

## Online Sampling

Host Computer




Randomly select accesses to monitor

**Address Stream**

1:read A
2:read B
3:read C
4:write C
5:read B
6:read D
7:read A
8:read E
9:read B

**Reuse Distance= 5**

**Reuse Distance=3**

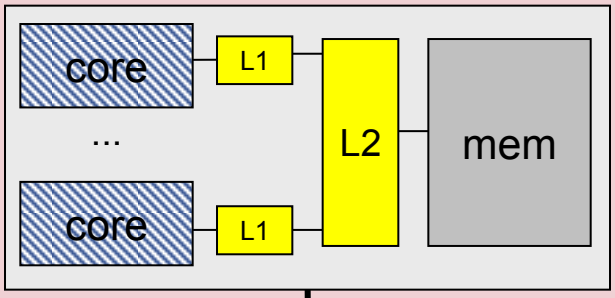
**Sparse Sampler**

**Application Fingerprint**

5, 3,…

## Offline "Insight Technology"

Target Architecture

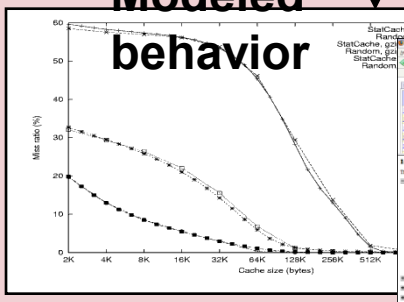



**Architectural Parameters**

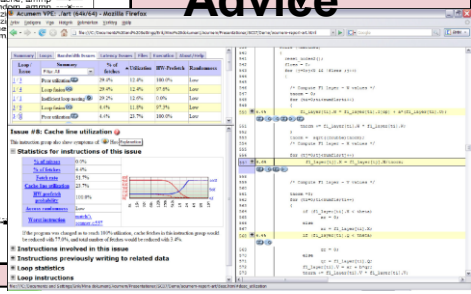
**Probabilistic Cache Model**

**ThreadSpotter**

**Modeled behavior**




**Advice**




Copyright 2012 Rogue Wave Software, Inc.

# ThreadSpotter - GUI

Acumem ThreadSpotter™

File

**Sample source**

- Sample application
  - Launch application
    - Program [ ] Browse...
    - Arguments [ ]
    - Working directory [/opt/acumem/threadspotter/bin] Browse...
  - Attach to running application
    - Pid [ ] Select...
  - Advanced sampling settings...
  - Sample application
- Read sample file
  - Sample file [/opt/acumem/threadspotter/bin/sample.s] Browse...

**Report generation**

- Generate report
  - Report file [report.tsr] Browse...
  - Processor model: Genuine Intel(R) CPU T2500 @ 2.00GHz Select...
  - Cache level: L2 (2 MB)
  - ☑ Launch report viewer
  - Advanced report settings...
- View existing report
  - Report file [/opt/acumem/threadspotter/bin/report.ts] Browse...

Sample application and generate report


Copyright 2012 Rogue Wave Software, Inc.

# Attach to a Process

Acumem ThreadSpotter™

File

**Sample source**

- Sample application
  - Launch application
    - Program
    - Arguments
    - Working directory /opt/acumem/th
  - Attach to running application
    - Pid
    - Advanc
    - S
  - Read sample file
    - Sample file /opt/acumem/th

**Report generation**

- Generate report
  - Report file report.tsr
  - Processor model: Genuine Intel(R
  - Cache level: L2 (2 MB)
  - Launch report viewer
    - Advar
- View existing report
  - Report file /opt/acumem/th

Sample applic

**Select process**

Select process to sample.

| PID | Name |
|---|---|
| 680 | /usr/lib/gvfs/gvfsd |
| 686 | /usr/lib/gvfs//gvfs-fuse-daemon |
| 1304 | /bin/sh |
| 1567 | /usr/bin/VBoxClient |
| 1576 | /usr/bin/VBoxClient |
| 1582 | /usr/bin/VBoxClient |
| 1993 | dbus-launch |
| 1996 | /bin/dbus-daemon |
| 2055 | kdeinit4: kdeinit4 Running... |
| 2072 | kdeinit4: klauncher [kdeinit] --fd=9 |
| 2121 | kdeinit4: kded4 [kdeinit] |
| 2174 | kwrapper4 |
| 2175 | kdeinit4: ksmserver [kdeinit] |
| 2177 | kwin |
| 2179 | kdeinit4: kglobalaccel [kdeinit] |
| 2183 | kdeinit4: plasma-desktop [kdeinit] |
| 2185 | /usr/bin/knotify4 |
| 2191 | kdeinit4: kaccess [kdeinit] |
| 2199 | kdeinit4: nepomukserver [kdeinit] |
| 2201 | kdeinit4: krunner [kdeinit] |
| 2204 | krandrtray |
| 2205 | /usr/lib/polkit-gnome/polkit-gnome-authentication... |

Aktualisieren | Abbrechen | OK


Copyright 2012 Rogue Wave Software, Inc.

# Advanced Sampling Settings

Advanced sampling settings

Start sampling

- Immediately
- After 1 seconds
- When function main has been called 1 times
- When address has been executed 1 times

Stop sampling

- When the program terminates
- After 1 seconds
- When function main has been called 1 times
- When address has been executed 1 times

Line sizes 64 Select...

Burst sampling, estimated run time 60 minutes normal quality

Initial sample period 100000 accesses

Safe stack handling

Save sample file as /opt/acumem/threadspot Browse...

OK


Copyright 2012 Rogue Wave Software, Inc.

# Advanced Report Settings

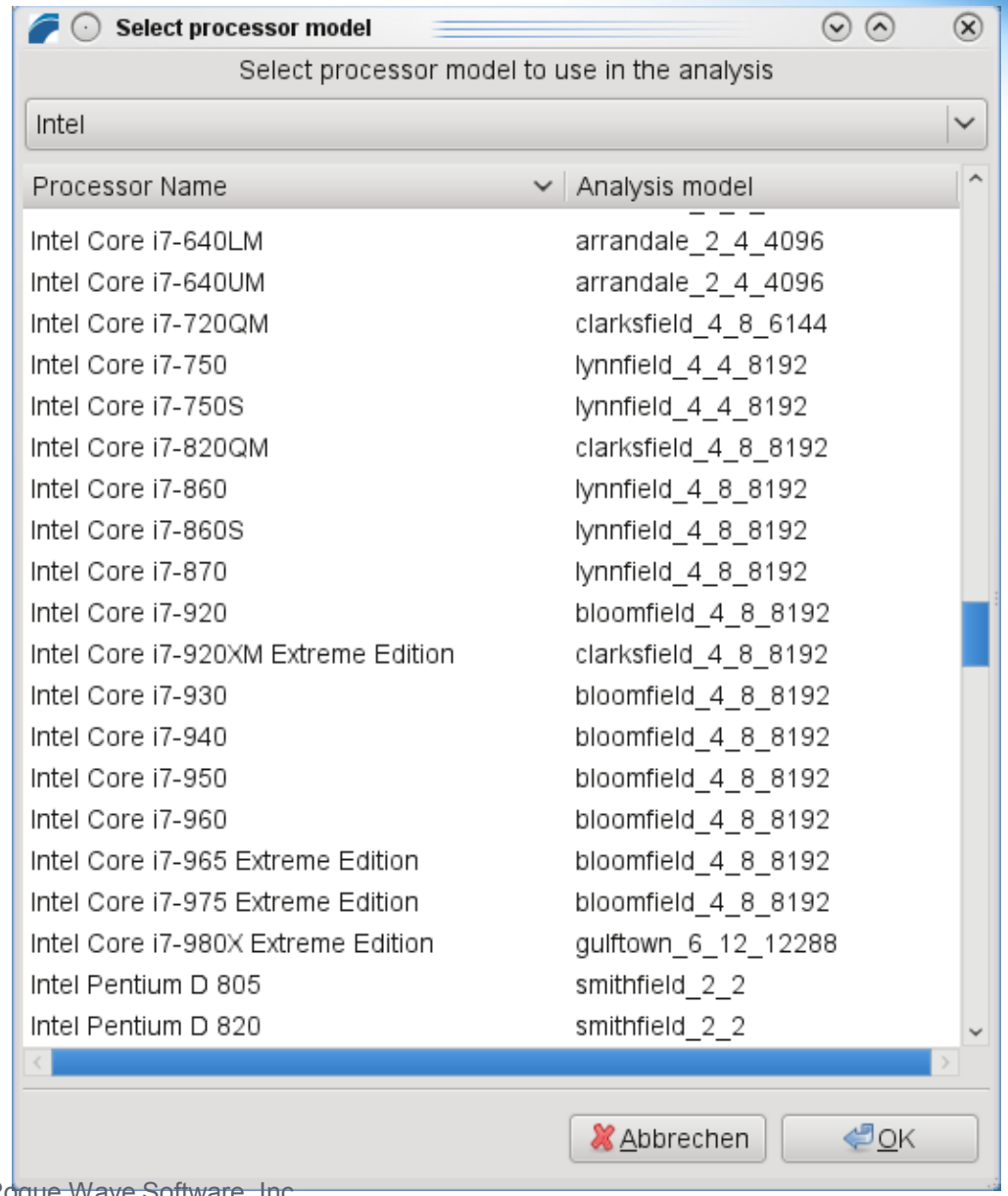
**Advanced report settings**

- [ ] Cache size — 64k — bytes
- [ ] Line size — 64 — bytes
- [ ] Number of caches — 0
- Replacement policy — Random
- Call stack depth — 1 — frames
- [ ] Read debug info from — Browse...
- [ ] Additional debug directories — Browse...
- [ ] Lookup source in — Browse...
- [ ] C++ symbol demangler — Browse...
- Ignore issue involving less than — 1 — % of all fetches
- [ ] Report title
- [ ] Verbose report generation output

OK


Copyright 2012 Rogue Wave Software, Inc.

# Processor Model Selection





Copyright 2012 Rogue Wave Software, Inc.

# Starting from Command Line

**Sampling:**

```
$ /opt/threadspotter/bin/sample -o <sample file name>.smp
  -r <program file and parameters>
```

**Creating a report:**

```
$ /opt/threadspotter/bin/report -i <sample file name>.smp
  -o <report file name>
```

**Viewing a report:**

```
$ /opt/threadspotter/bin/view -i <report file name>
```


Copyright 2012 Rogue Wave Software, Inc.

# ThreadSpotter - Report

Acumem ThreadSpotter: tripleissue (2M/64) - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:59311/session/front.html

Meistbesuchte Seiten | openSUSE | Getting Started | Latest Headlines | Mozilla Firefox

Acumem ThreadSpotter: tripleissue ...

**Acumem ThreadSpotter™**

Acumem ThreadSpotter™ is a tool to quickly analyze an application for a range of performance problems, particularly related to multicore optimization.

Read more... Manual

Open the Report

**Your application**

**Application:** ./tripleissue

**Memory Bandwidth**

The memory bus transports data between the main memory and the processor. The capacity of the memory bus is limited. Abuse of this resource limits application scalability.
Manual: Bandwidth

**Memory Latency**

The regularity of the application's memory accesses affects the efficiency of the hardware prefetcher. Irregular accesses causes cache misses, which forces the processor to wait a lot for data to arrive.
Manual: Cache misses Manual: Prefetching

**Data Locality**

Failure to pay attention to data locality has several negative effects. Caches will be filled with unused data, and the memory bandwidth will waste transporting unused data.
Manual: Locality

**Thread Communication / Interaction**

Several threads contending over ownership of data in their respective caches causes the different processor cores to stall.
Manual: Multithreading

This means that your application shows opportunities to:

**Tune cache utilization to avoid processor stalls.**

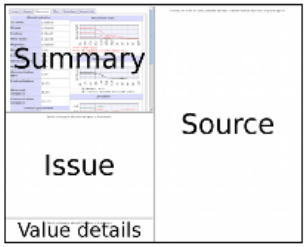
Read more...

**Next Steps**

The prepared report is divided into sections.

- Select the tab **Summary** to see global statistics for the entire application.
- Select the tabs **Bandwidth Issues**, **Latency Issues** and **MT Issues** to browse through the detected problems.
- Select the tab **Loops** to browse through statistics and detected problems loop by loop.

The Issue and Source windows contain details and annotated source code for the detected problems.




**Resources**

**Manual**

Table of Contents | Overview
Optimization Workflow | Concepts
Reading the Report | Issue Reference

**Rogue Wave Software Web Site**

Rogue Wave Web Site | Tutorials

Done

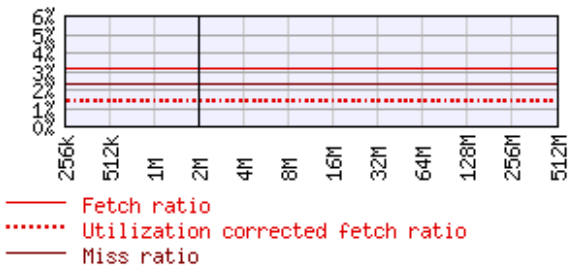

Copyright 2012 Rogue Wave Software, Inc.

# Summary

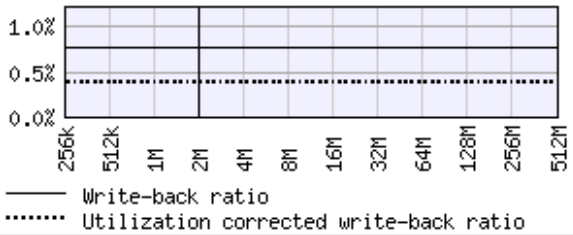
Issues | Loops | Summary | Files | Execution | About/Help

| Global statistics | |
|---|---|
| Accesses | 6.14e+07 |
| Misses | 1.48e+06 |
| Fetches | 1.96e+06 |
| Write-backs | 4.72e+05 |
| Upgrades | 0.00e+00 |
| Miss ratio | 2.4% |
| Fetch ratio | 3.2% |
| Write-back ratio | 0.8% |
| Upgrade ratio | 0.0% |
| Communication ratio | 0.0% |
| Fetch utilization | 43.7% |
| Write-back utilization | 52.0% |
| Communication utilization | 100.0% |
| **Analysis parameters** | |
| Processor model | Genuine Intel(R) CPU T2500 @ 2.00GHz (auto) |
| Number of CPUs | 1 |
| Number of caches | 1 |
| Cache level | 2 |
| Cache size | 2M |
| Line size | 64 |
| Replacement policy | random |
| Software prefetches active | Yes |

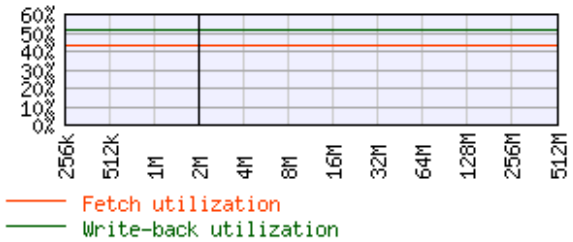
Miss/Fetch ratio




Write-back ratio




Utilization

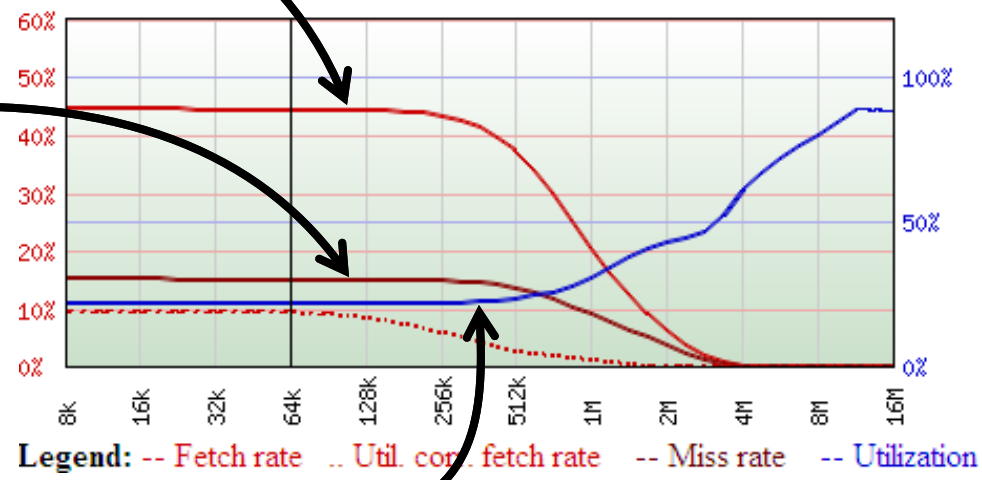




Copyright 2012 Rogue Wave Software, Inc.

# Metrics as a function of cache size

- Fetch ratio
  - The likelihood that a memory access causes memory bus traffic
- Miss ratio
  - The likelihood that a memory access doesn't find requested data in the cache
- Fetch utilization
  - How much of every fetched cache line that the application really uses





Copyright 2012 Rogue Wave Software, Inc.

# Issues by Severity

Issues | Loops | Summary | Files | Execution | About/Help

Bandwidth Issues | Latency Issues | Multi-Threading Issues | Pollution Issues

| # | | Issue type<br>Filter: All | % of bandwidth | % of fetches | % of write-backs | Fetch utilization | Write-back utilization |
|---|---|---|---|---|---|---|---|
| 8 | F | Fetch utilization | 50.0% | 49.8% | 50.8% | 50.0% | 51.0% |
| 9 | W | Write back utilization | 50.0% | 49.8% | 50.8% | 50.0% | 51.0% |
| 6 | F | Fetch utilization | 29.4% | 24.7% | 49.2% | 25.1% | 53.0% |
| 7 | W | Write back utilization | 29.4% | 24.7% | 49.2% | 25.1% | 53.0% |
| 10 | F | Fetch utilization | 20.5% | 25.5% | 0.0% | 49.2% | 100.0% |

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

## Issue #8: Fetch utilization

This instruction group also show symptoms of: Fetch hot-spot, Write-back hot-spot.

+ **Statistics for instructions of this issue**
+ **Instructions involved in this issue**
+ **Instructions previously writing to related data**
+ **Loop statistics**
+ **Loop instructions**

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.


Copyright 2012 Rogue Wave Software, Inc.

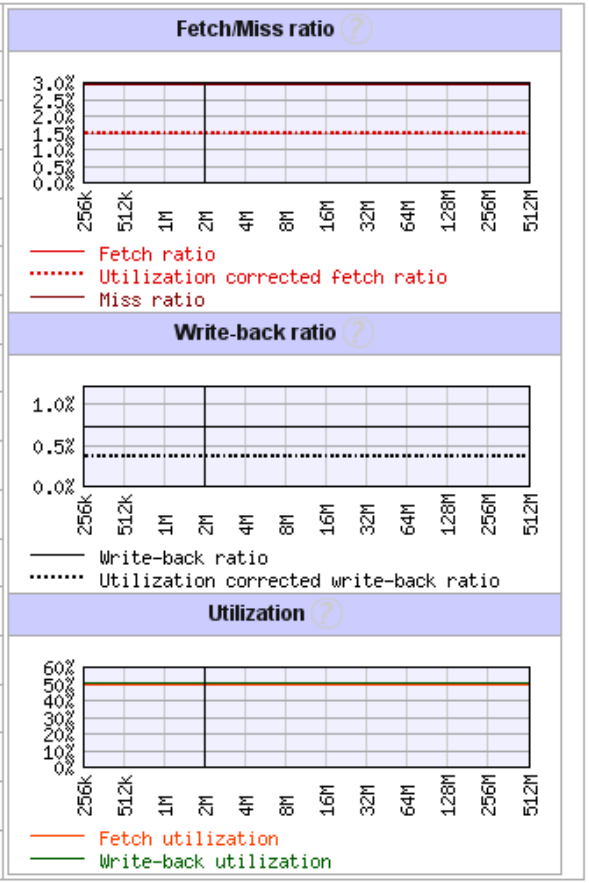
# Statistics of an Issue

Issues | Loops | Summary | Files | Execution | About/Help

Bandwidth Issues | Latency Issues | Multi-Threading Issues | Pollution Issues

## Statistics for instructions of this issue

| | |
|---|---|
| Accesses | 3.27e+07 |
| % of misses | 66.1% |
| % of bandwidth | 50.0% |
| % of fetches | 49.8% |
| % of write-backs | 50.8% |
| % of upgrades | --- |
| Miss ratio | 3.0% |
| Fetch ratio | 3.0% |
| Write-back ratio | 0.7% |
| Upgrade ratio | 0.0% |
| Communication ratio | 0.0% |
| Fetch utilization | 50.0% |
| Write-back utilization | 51.0% |
| Communication utilization | 100.0% |
| False sharing ratio | 0.0% |
| HW prefetch probability | 0.0% |
| Access randomness | Low |
| Worst instruction | tripleissue!main()+0x6f (0x80490ad) [R], tripleissue.cpp:53 |




If the program was changed as to reach 100% fetch utilization, fetches in this instruction group would be reduced with 50.0%, and total number of fetches would be reduced with 24.9%.


Copyright 2012 Rogue Wave Software, Inc.

# Reference to Source Code

```
45                  //Partially used structure
46
47                  TS_sampler_start(200);
48
49                  pMyData = new DATA[10*1024*1024];
50
51  24.7%           for (long i=0; i<10*1024*1024; i++)
52                  {
53  75.3%                   pMyData[i].a = pMyData[i].b;
54                  }
55
56                  delete pMyData;
57
58                  TS_sampler_stop();
59
60
61
62                  //Inefficient loop nesting
63
64                  char * p = new char[SIZE];
65
66                  long nbRows = NBROWS;
67                  long sRowSize = ROWSIZE;
68
```

| % of fetches | Miss ratio | Fetch ratio | WB ratio | Fetch Util | WB Util | PC | Type | Issues |
|---|---|---|---|---|---|---|---|---|
| 14.1% | 3.4% | 3.4% | 0.0% | 49.2% | 100.0% | 0x8049096 | R | F |
| 12.5% | 3.0% | 3.0% | 0.0% | 50.0% | 51.0% | 0x804909b | R | F W |
| 11.4% | 2.7% | 2.7% | 0.0% | 49.2% | 100.0% | 0x80490a8 | R | F |
| 12.7% | 3.0% | 3.0% | 0.0% | 50.0% | 51.0% | 0x80490ad | R | F W |
| 12.8% | 0.0% | 4.1% | 0.0% | 25.1% | 53.0% | 0x80490b9 | R | F W |
| 11.9% | 0.0% | 3.8% | 3.8% | 25.1% | 53.0% | 0x80490bc | W | F W |


Copyright 2012 Rogue Wave Software, Inc.

# Used Icons

## Slowspot Issues

- Fetch utilization
- Write back utilization
- Communication utilization
- Inefficient loop nesting
- Random access
- Prefetch: too close
- Prefetch: too distant
- Prefetch: unnecessary
- False sharing

## Opportunity issues

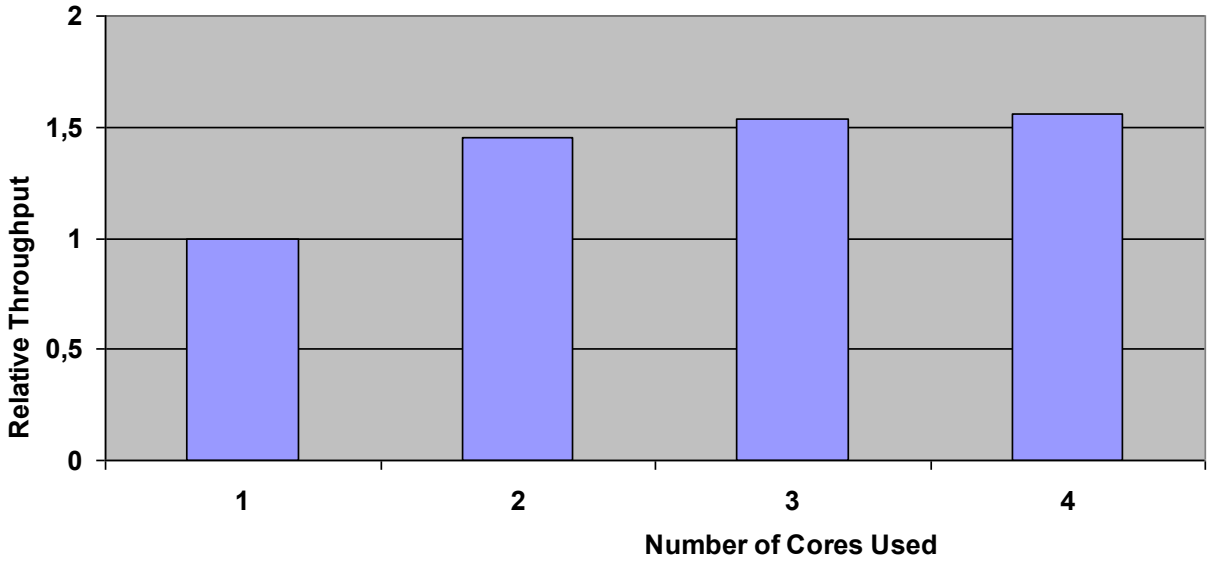
- Spatial blocking
- Temporal blocking
- Spat/temp blocking
- Loop fusion
- Non-temporal data
- Non-temporal store possible
- Fetch hot-spot
- Write-back hot-spot
- Communication hot-spot


Copyright 2012 Rogue Wave Software, Inc.

# Resource Sharing Example

**Libquantum**
A quantum computer simulation
Widely used in research (download from: http://www.libquantum.de/ )
4000+ lines of C, fairly complex code.
Runs an experiment in ~30 min

Throughput improvement:





Copyright 2012 Rogue Wave Software, Inc.

# Demo

Applications Places System 1.83 GHz 2:16 PM Swe

Computer
erik's Home
Trash
usr
koko on 192.168.244.1

**Acumem SlowSpotter™**

File

**Sample source**

Sample application

Launch application

Program ./libquantum Browse...

Arguments 1397 8

Working directory /home/erik/demos/libqu Browse...

Attach to running application

Pid Select...

Advanced sampling settings...

Sample application

Read sample file

Sample file /home/erik/sample.smp Browse...

**Report generation**

Generate report in /home/erik/Reports Browse...

Report name LQ_orig_stack0

Cache size 2M bytes

Launch web browser /usr/bin/htmlview Browse...

Advanced report settings...

Sample application and generate report

**Libquantum:**

Orig code

Spatial opt

Spat + Loop fusion

Edit-compile-analysis cycle ≈ 1min

[erik@localhost... [koko on 192.1... [Acumem Slow... [erik] [demos] Acumem SlowS... [file:// - 8.4. Loo... Starting Take Scr...


Copyright 2012 Rogue Wave Software, Inc.

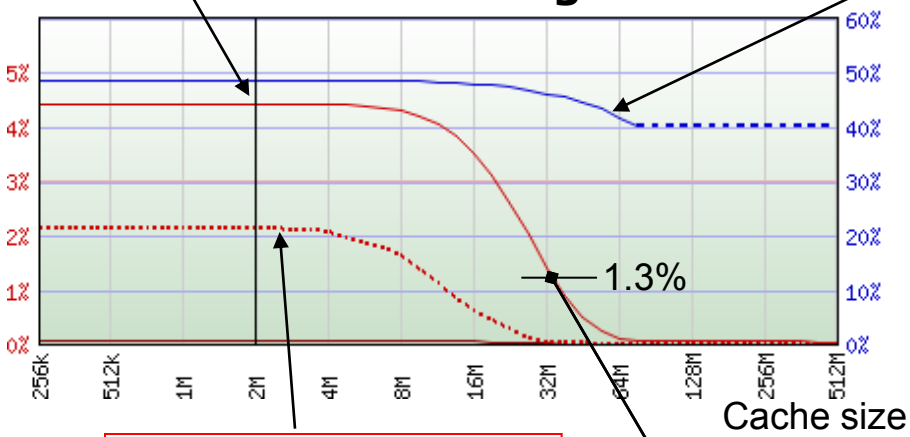
# Utilization Analysis

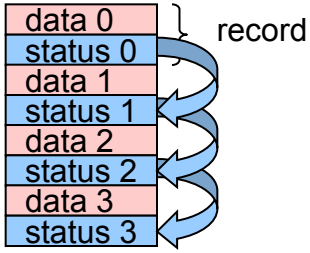
**Libquantum**

Cache utilization ≈ Fraction of cache data utilized

Fetch rate

**Original Code**




Only accessing status data in main loop

Predicted fetch rate if utilization = 100%

**Need 32 MB per thread!**


Copyright 2012 Rogue Wave Software, Inc.

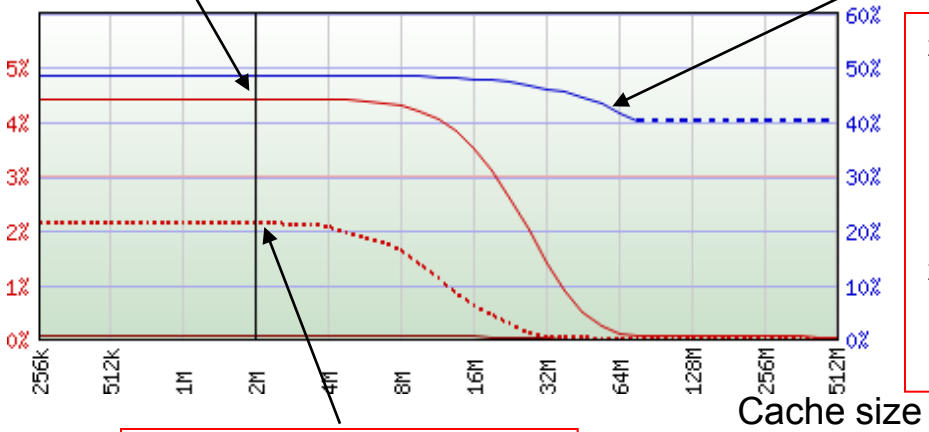
# Utilization Analysis

**Libquantum**

Cache utilization ≈ Fraction of cache data utilized

Fetch rate

**Original Code**




Predicted fetch rate if utilization = 100%

**Utilization Optimization**

```
for (i=0; i++; i<MAX) {
    ... = huge_data[i].status + ...
 }
```

```
for (i=0; i++; i<MAX) {
    ... = huge_data_status[i] + ...
 }
```

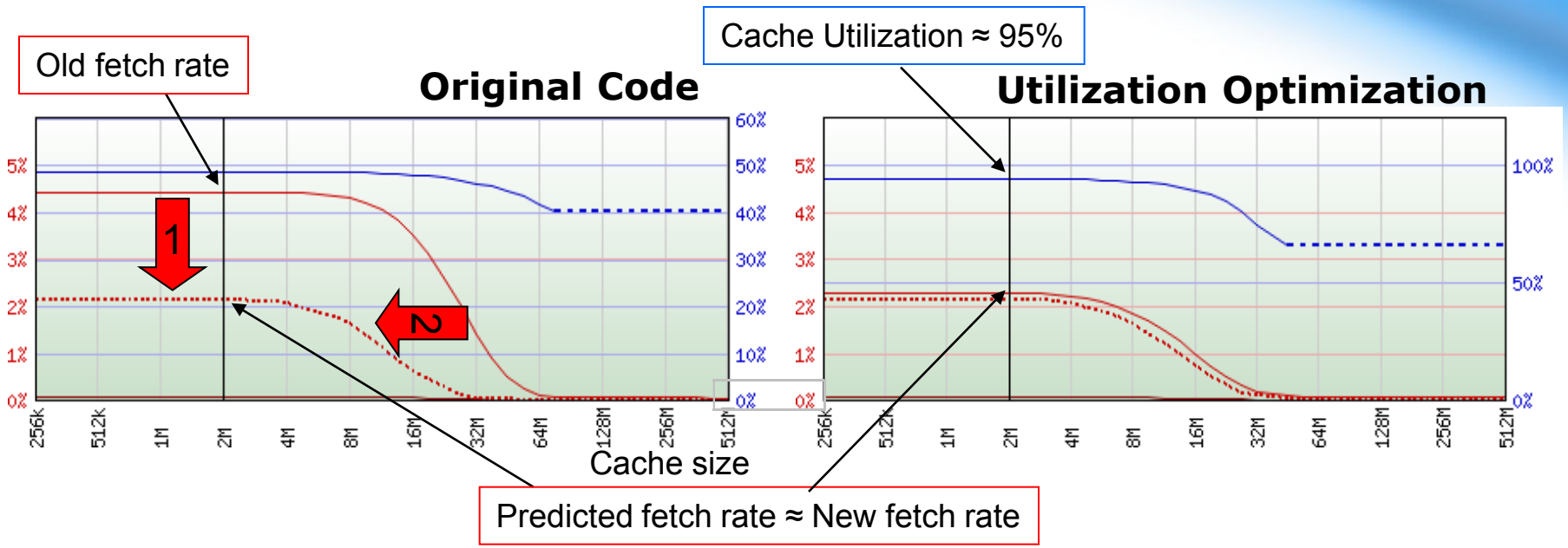
## ThreadSpotter's First Advice: Improve Utilization

➔Change one data structure

- Involves ~20 lines of code
- Takes a non-expert 30 min


Copyright 2012 Rogue Wave Software, Inc.

# Utilization Optimization



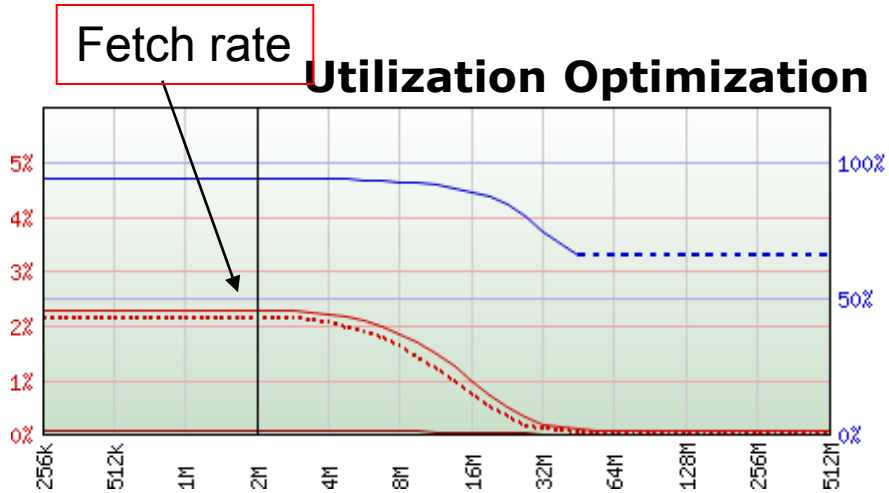

**Two positive effects from better utilization**

1. Each fetch brings in more useful data → lower fetch rate
2. The same amount of useful data can fit in a smaller cache → shift left


Copyright 2012 Rogue Wave Software, Inc.

# Loop Fusion

**Libquantum**




**Utilization + Fusion Optimization**

```
...
toffoli(huge_data, ...)
cnot(huge_data, ...
...

...
fused_toffoli_cnot(huge_data,...)
...
```

Second-Fifth ThreadSpotter Advice: Improve reuse of data through loop fusion

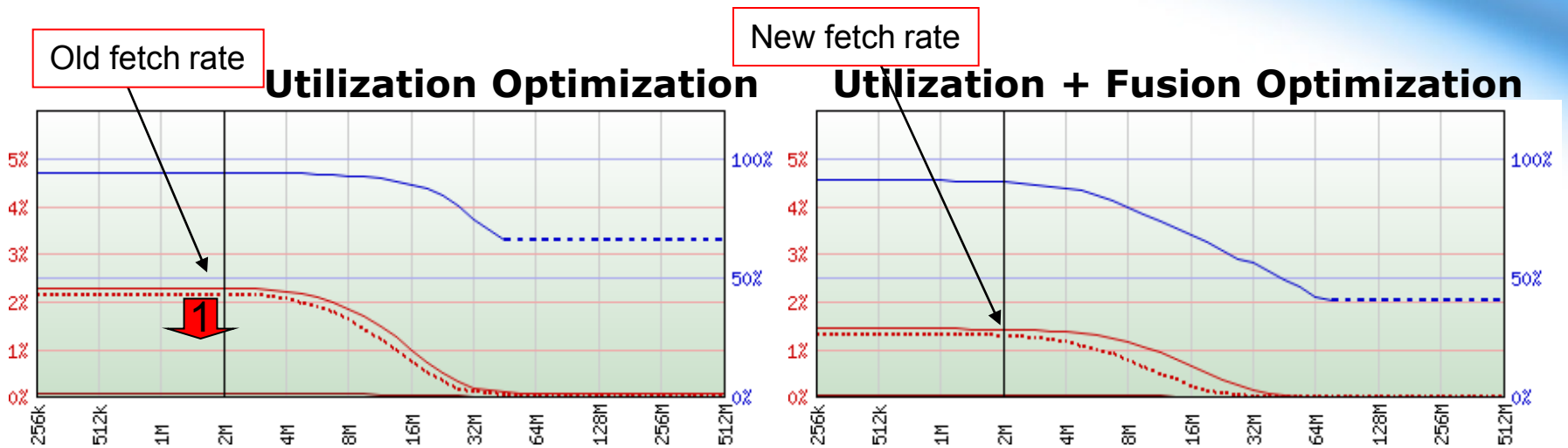
➔**Fuse functions traversing the same data**

- **Here: four fused functions created**
- **Takes a non-expert < 2h**


Copyright 2012 Rogue Wave Software, Inc.

# Effect: Loop Fusion

**SPEC CPU2006-462.libquantum**

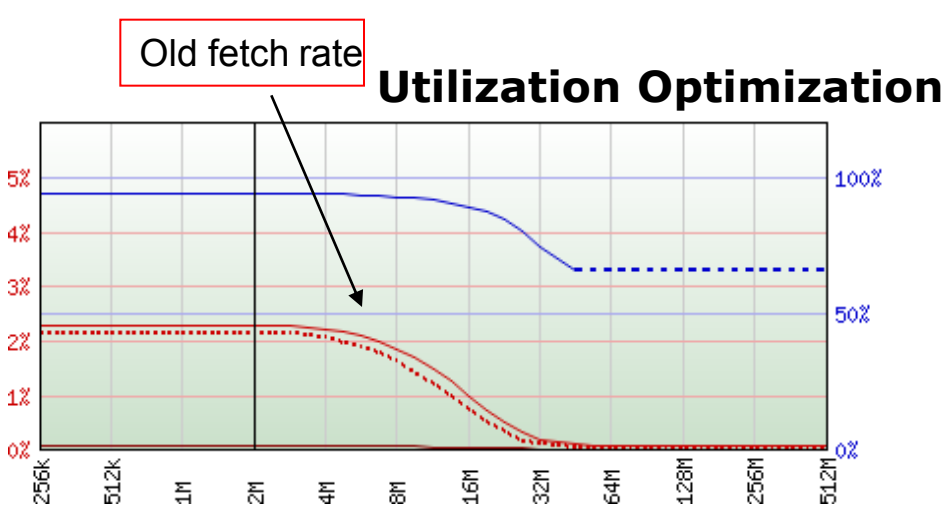


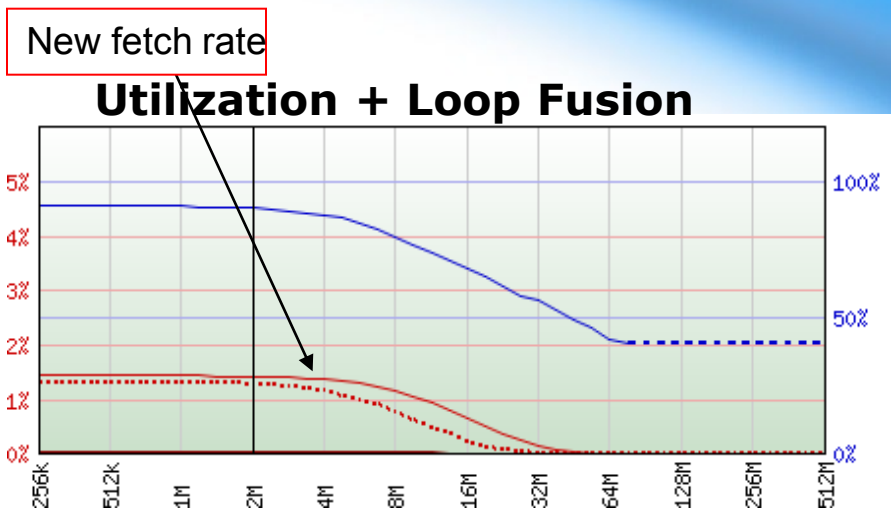

- The miss in the second loop goes away
- Still need the same amount of cache to fit "all data"


Copyright 2012 Rogue Wave Software, Inc.

# Utilization + Loop Fusion

**Libquantum**

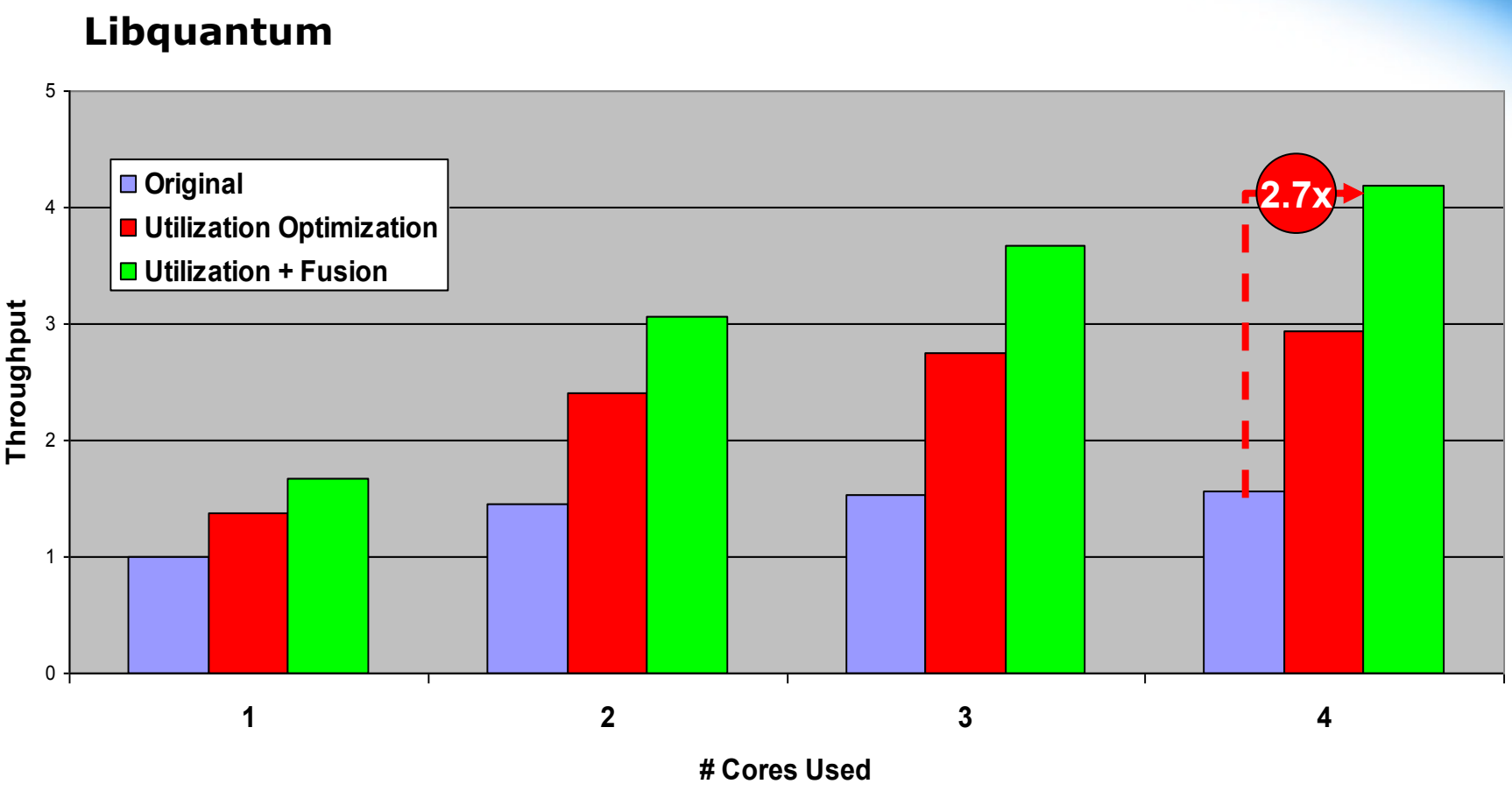



- **Fetch rate down to 1.3% for 2MB**
- **Same as a 32 MB cache originally**


Copyright 2012 Rogue Wave Software, Inc.

# Summary





Copyright 2012 Rogue Wave Software, Inc.

# Another Demo – N-body

 Copyright 2012 Rogue Wave Software, Inc.

# Simulation of Gravitational N-body Problem

- Initialize bodies
- for time = start to end step by Δt
  - calculate forces
  - move bodies
- end for time

for each body i=1 to n

$d\vec{v} = \vec{f}_i / m_i \times \Delta t$

$d\vec{p} = (\vec{v}_i + d\vec{v}/2) \times \Delta t$

$\vec{v}_i += d\vec{v}$

$\vec{p}_i += d\vec{p}$

$\vec{f}_i = \vec{0}$

end for each

for each body i=1 to n-1

for each neighbour j=i+1 to n

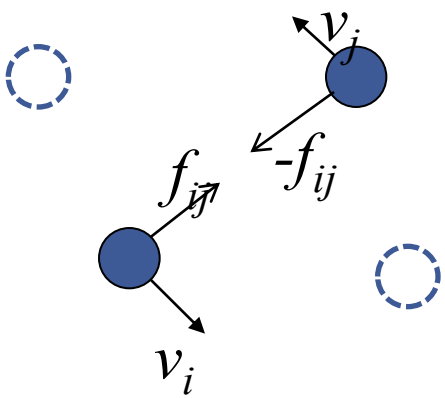
calculate $\vec{f}_{ij}$

$\vec{f}_i += \vec{f}_{ij}$

$\vec{f}_j -= \vec{f}_{ij}$

end for j

end for i





Copyright 2012 Rogue Wave Software, Inc.

# Algorithm

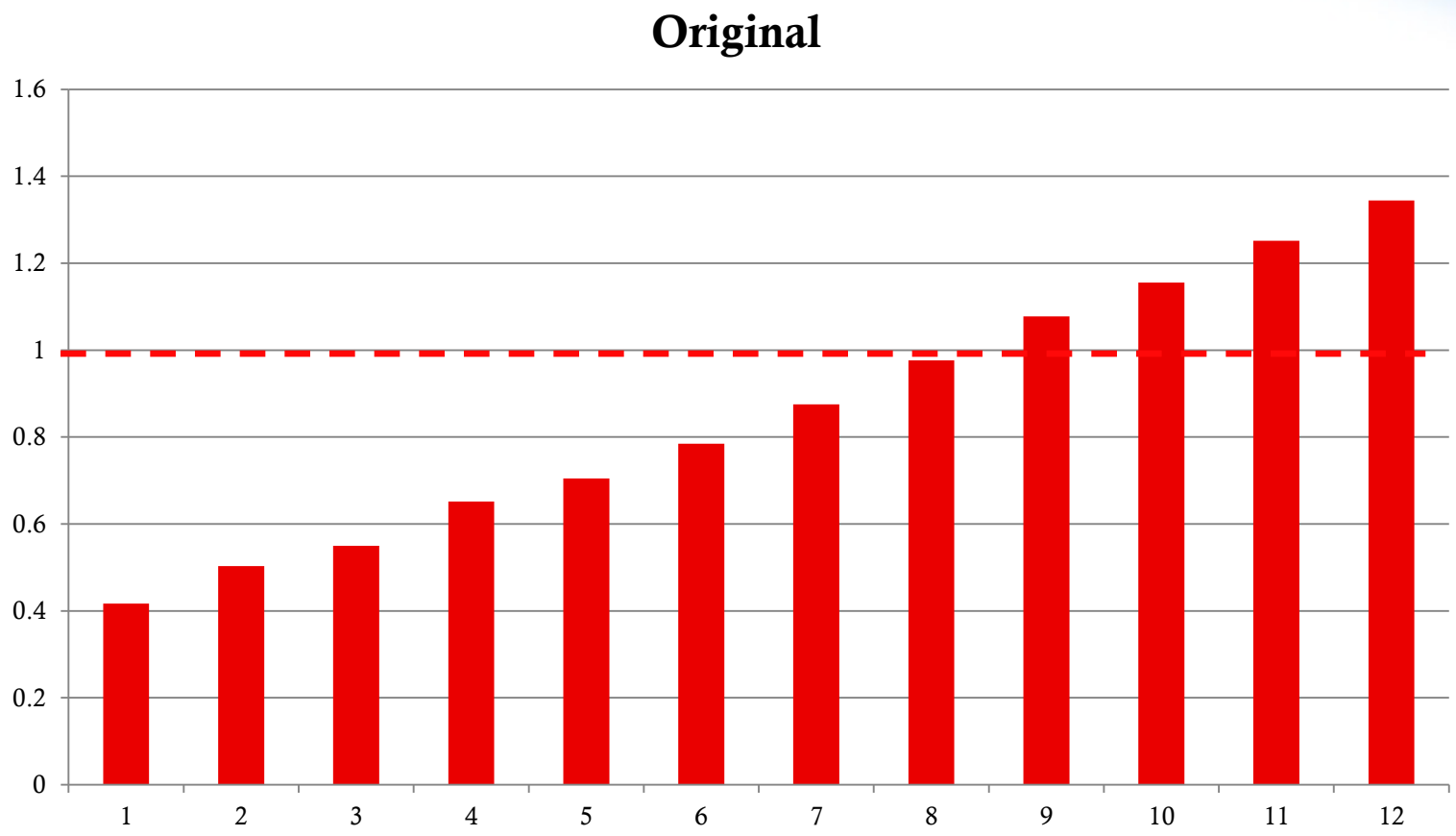
```
#pragma omp parallel private(i,j)
#pragma omp for
for (time=start to end, step dt)
{
    for(i=0 to n, step 1)
        for(j=i+1 to n, step 1)
        CalculateForce(bodyArr[i], bodyArr[j]);
    #pragma omp for
    for(i=0 to n, step 1)
        Move(bodyArr[i]);
}
```

```
typedef struct
{
    double px,py;
    double vx,vy;
    double fx,fy;
    double m;
} body;
```

```
#pragma omp atomic
force updates
```

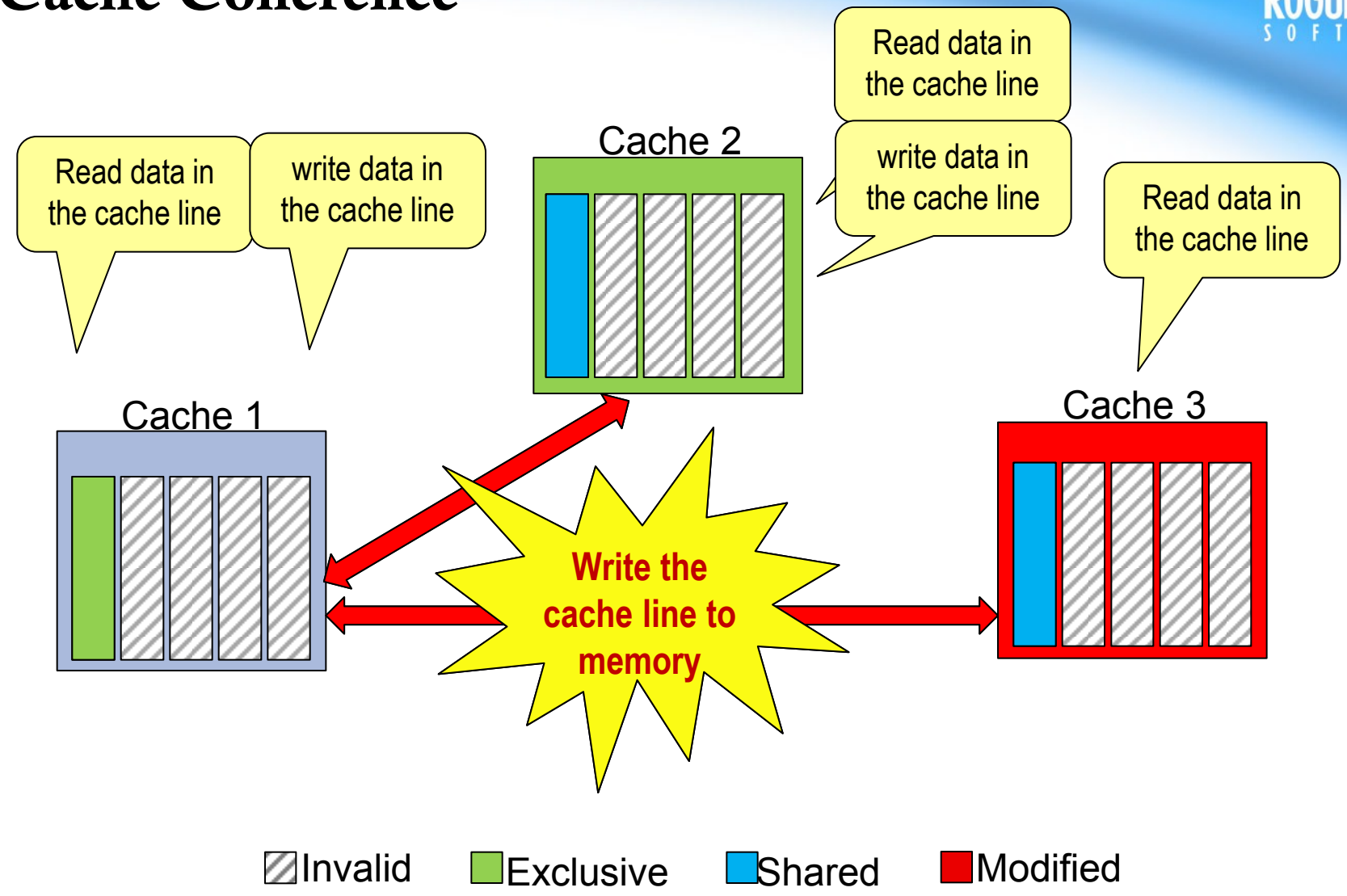

Copyright 2012 Rogue Wave Software, Inc.

# Speed Up or Slow Down?




 Copyright 2012 Rogue Wave Software, Inc.

# Cache Coherence



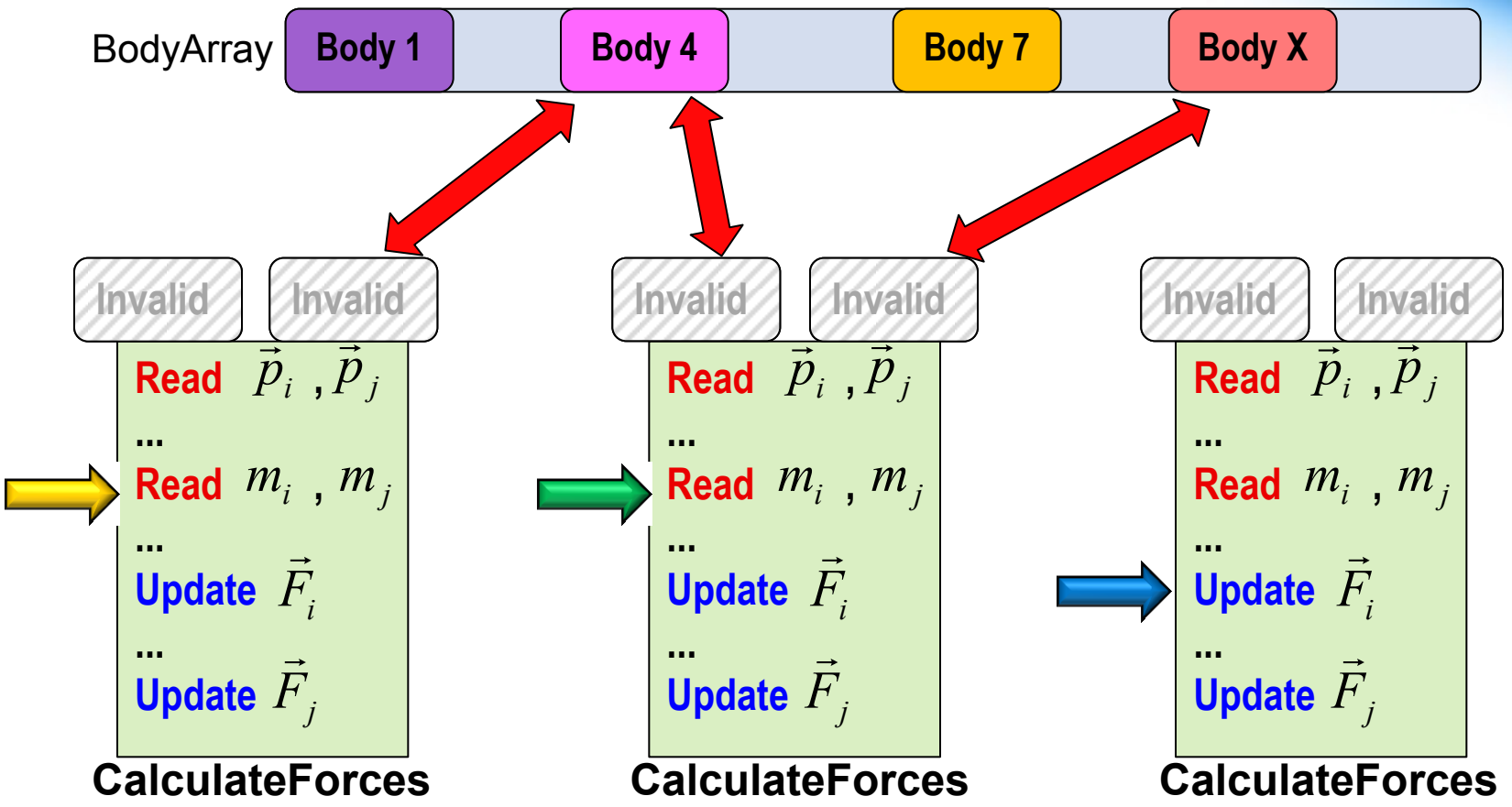

 Copyright 2012 Rogue Wave Software, Inc.

# Communication Overheads in Force Calculations

- Symmetric updates to the ’force’ vector causing false sharing:
  - Fighting over ownership of the corresponding cachelines.
  - Negative side-effect: No fast access to read-only variable ’position’.
- Low write-back utilization:
  - Dirty cache lines are written back to memory before re-updating force fields.
- Expected communication overheads due to atomic updates.


Copyright 2012 Rogue Wave Software, Inc.

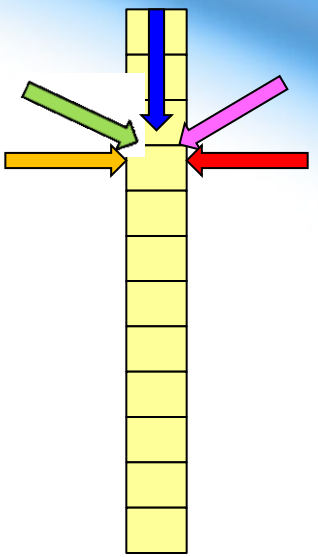
# Communication Overheads in Force Calculations



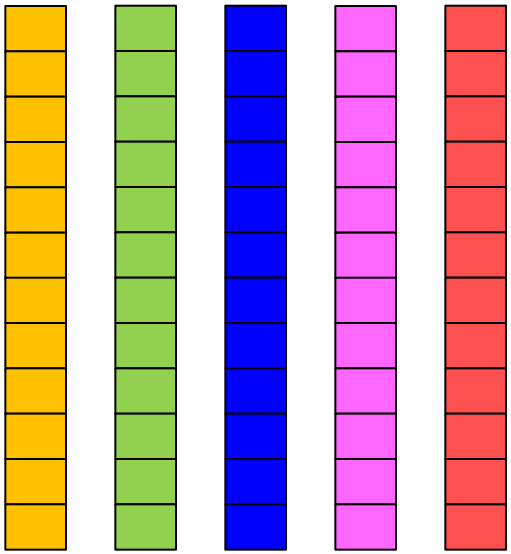

 Copyright 2012 Rogue Wave Software, Inc.

# Avoid False Sharing

```
typedef struct
{
        double px,py;
        double vx,vy;
        double fx,fy;
        double m;
} body;
```



Avoid atomic updates using thread private force buffer!




Copyright 2012 Rogue Wave Software, Inc.

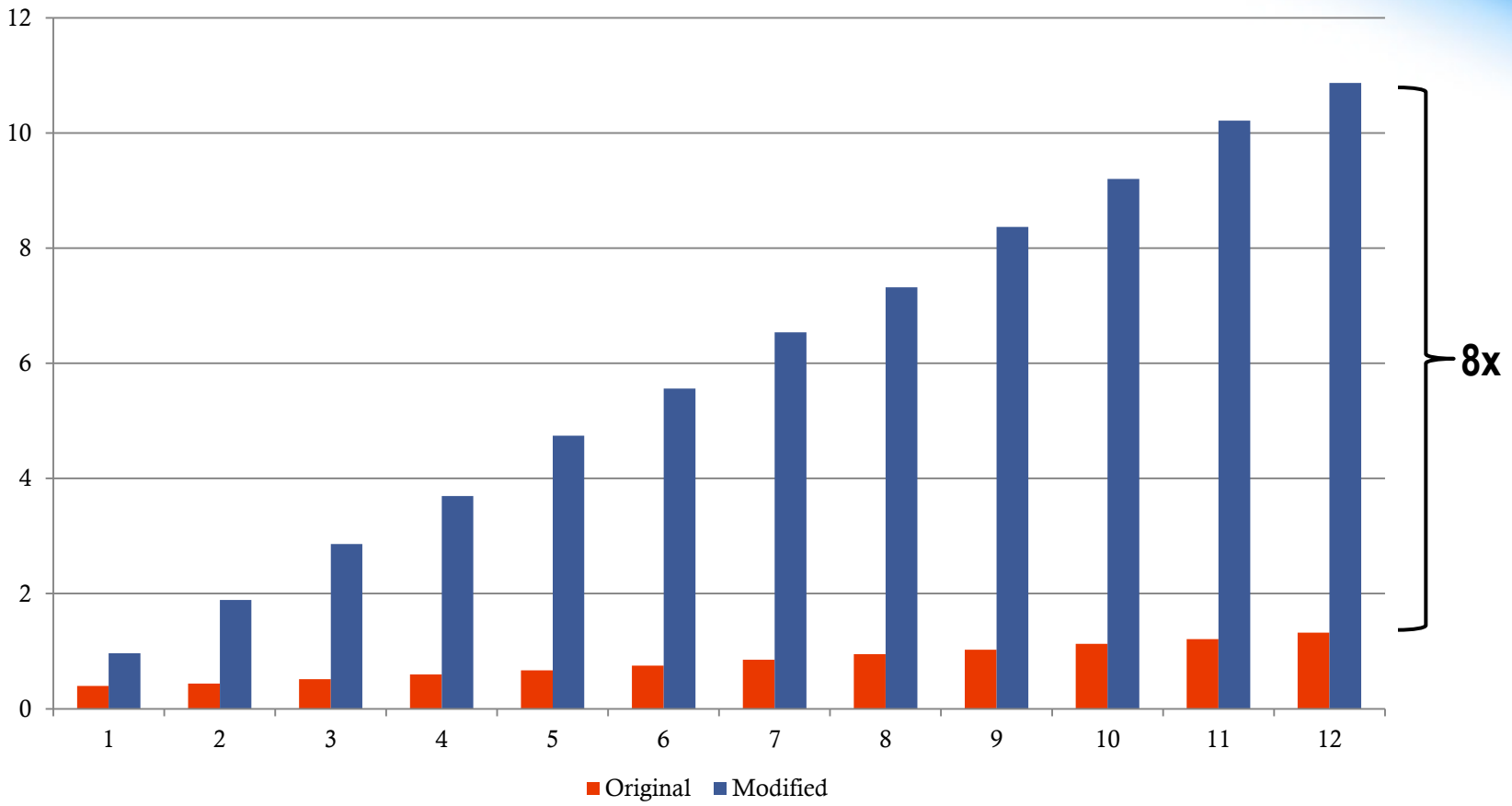
# Modified Algorithm

```
#pragma omp parallel private(i, j, id)
{
  id=omp_get_thread_num();
  for (time=start to end, step dt)
   {      forceArr[id, 0 to n] = 0
          for (i=id to n, step nThreads)      //Now able to scatter for load balancing
             for (j=i+1 to n, step 1)
                CalculateForce (bodyArr[i], bodyArr[j], forceArr[id,i], forceArr[id,j] );
         #pragma omp barrier
          for (i=id to n, step nThreads)      //move objects
                SumForcesAndMove (bodyArr[i], forceArr, i, nThr, n);
         #pragma omp barrier
   }
}
```

bodyForce **f** = **sum** forceArr[ 0 to nThreads, i]
**Move** bodyArr[i] using **f**


Copyright 2012 Rogue Wave Software, Inc.

# Overall Performance




 Copyright 2012 Rogue Wave Software, Inc.

# Questions ?

 Copyright 2012 Rogue Wave Software, Inc.

# Thank you