Xeon Phi Processor Tuning (2)

Kevin Olson Ning Li

May 9, 2017

nag®

Experts in numerical algorithms
and HPC services

CACHE Optimization

Introduction

- Tuning part 2 continues theme of memory
- Processors have features to cope with memory bottleneck
- MCDRAM was a large capacity high-bandwidth solution

Introduction

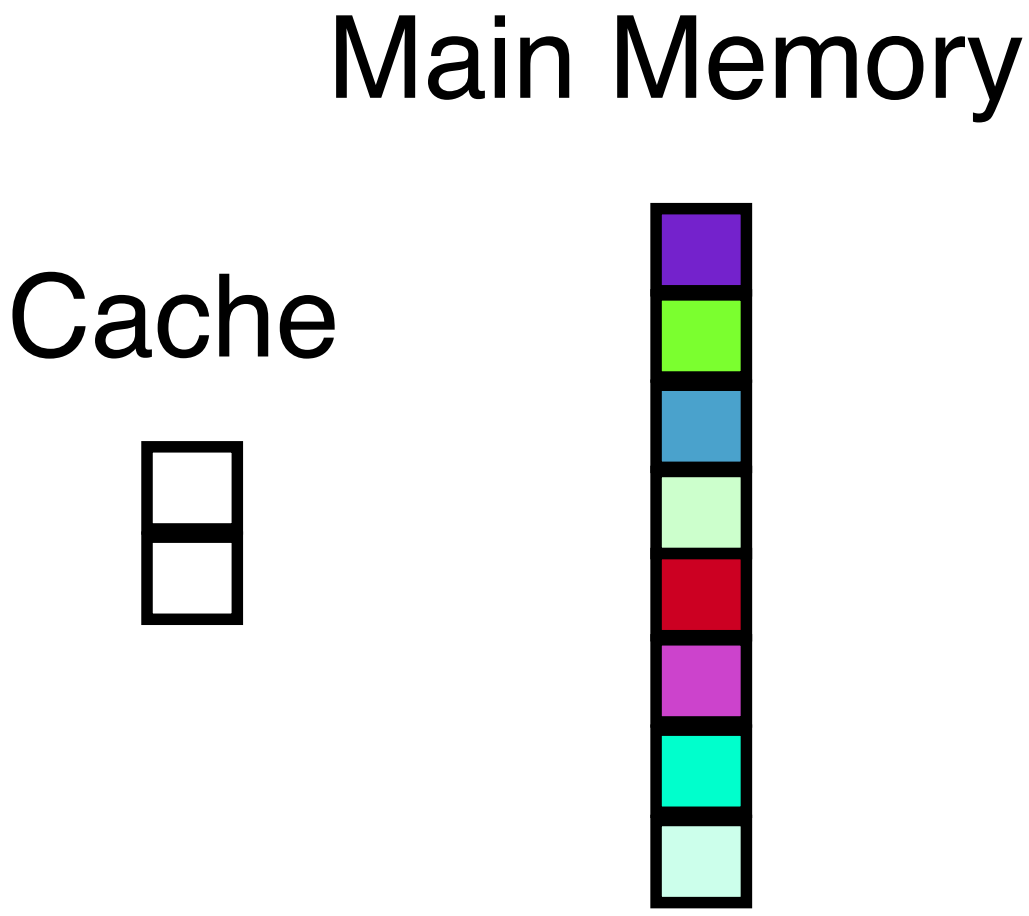
- There is also a low-capacity solution called Cache (NOT Cache mode we discussed earlier)
- Cache Is fast, but limited memory "close" to the processor.
- Cache is usually very limited (ranging from <1MB to 40MB, because its expensive).
- Unlike MCDRAM, very little control over how it works

Cache

```
for(int i=0;i<n;i++){
  for(int j=0;j<n;j++){
    a[i*n+j]+=1.0;
  }
}
```

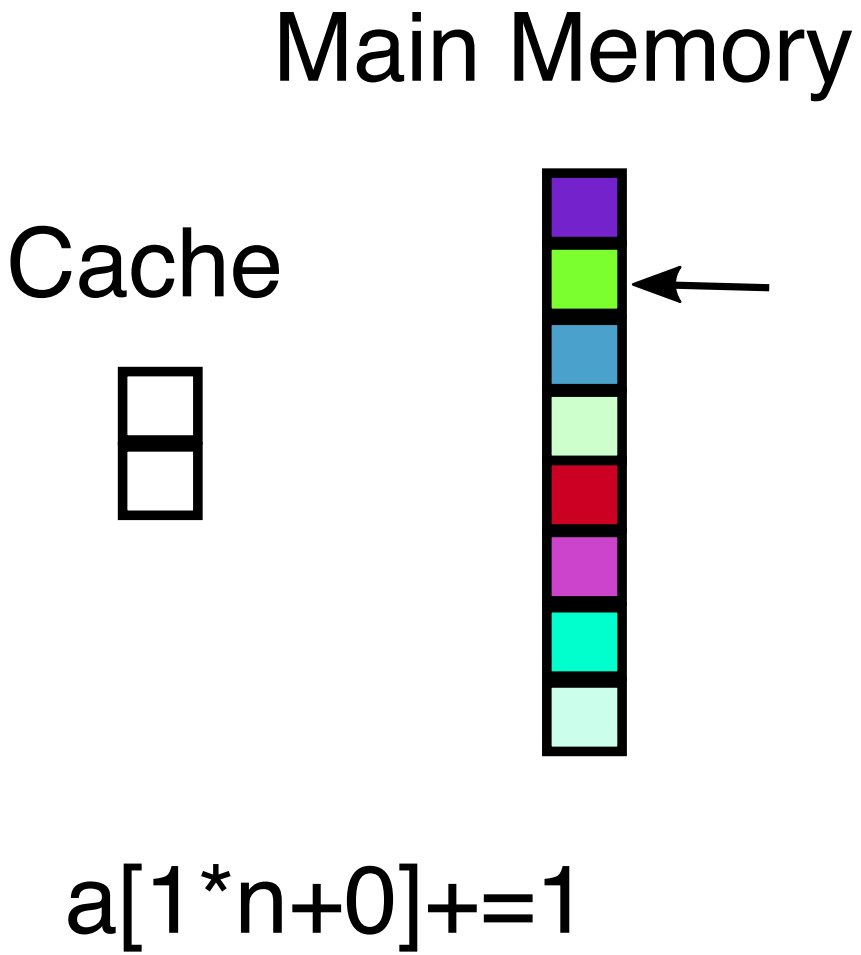
Cache: A Simple Model

Cache is small, main memory is large.

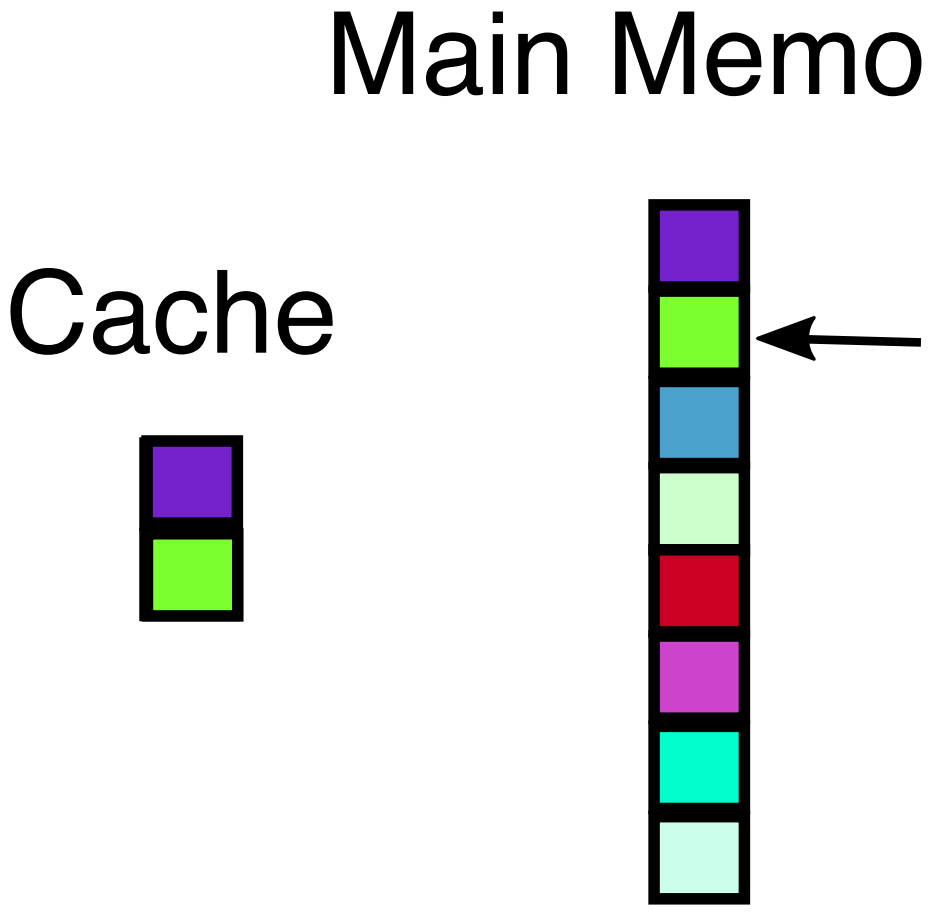


Cache: A Simple Model

Data must go from main memory through cache.



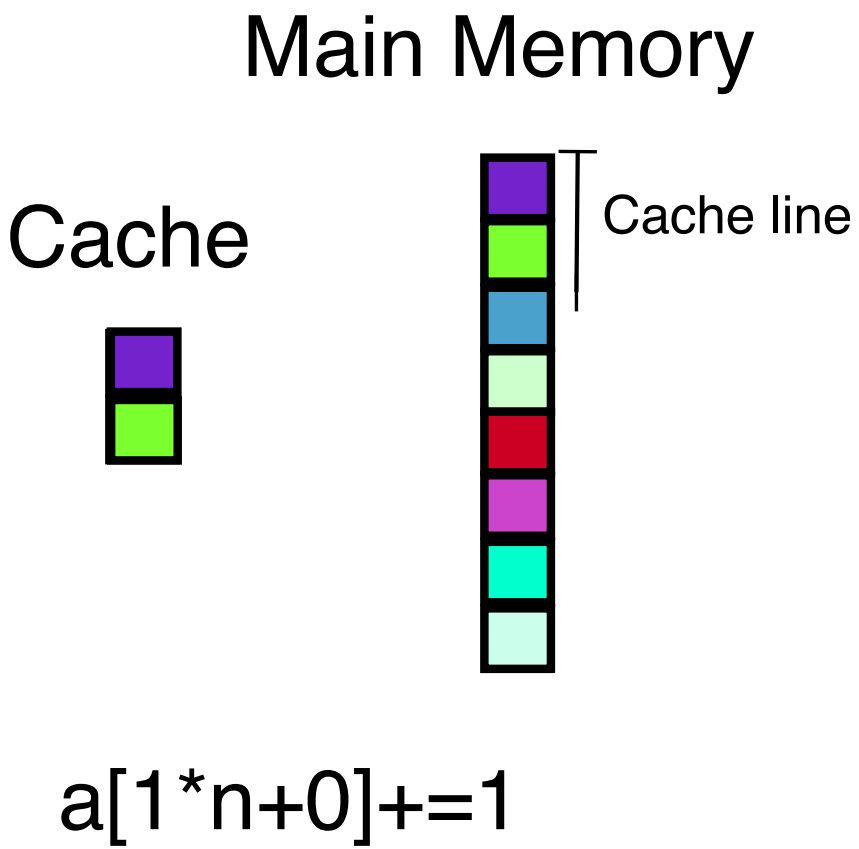
Cache: A Simple Model



a[1*n+0]+=1

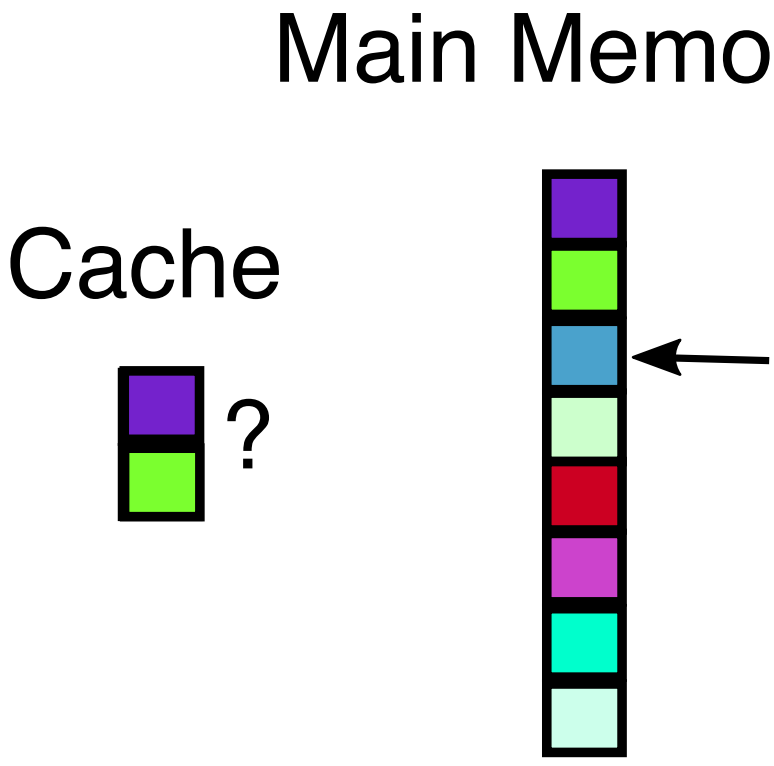
Cache: A Simple Model

in "lines", i.e. a set of contiguous memory locations that begin on byte boundaries.



Cache: A Simple Model

if a data is not in cache, we get a cache "miss"...

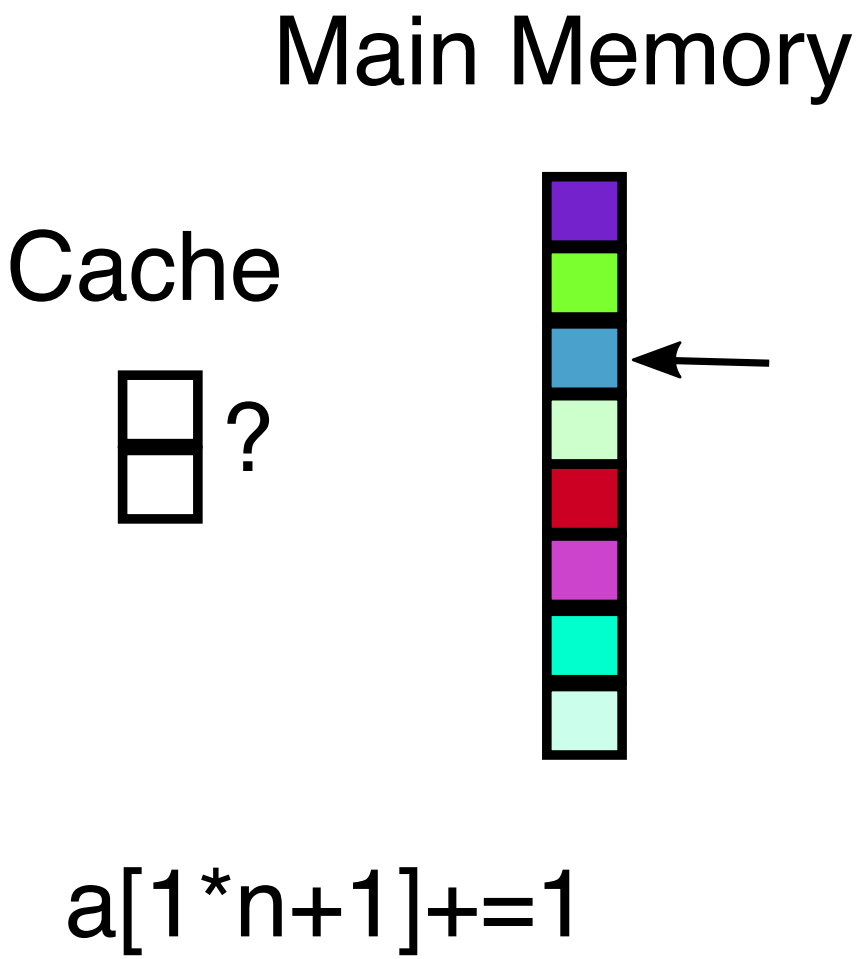


a[1*n+1]+=1

and a new cache line is loaded.

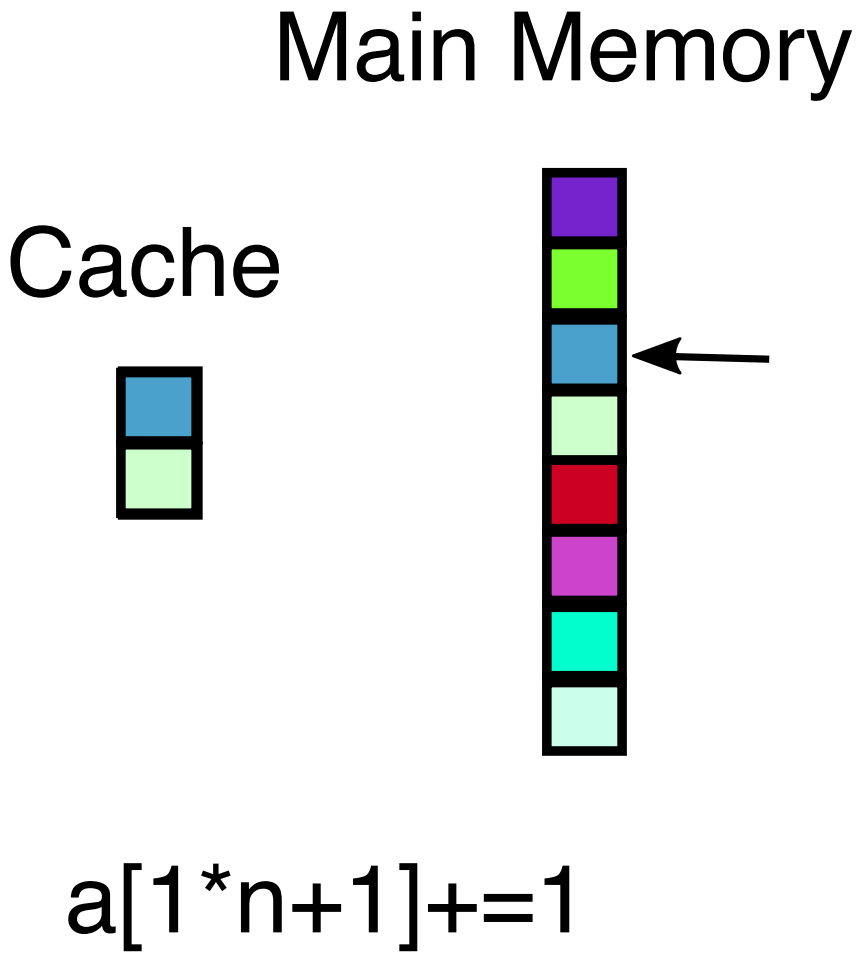
Cache: A Simple Model

cache is flushed...



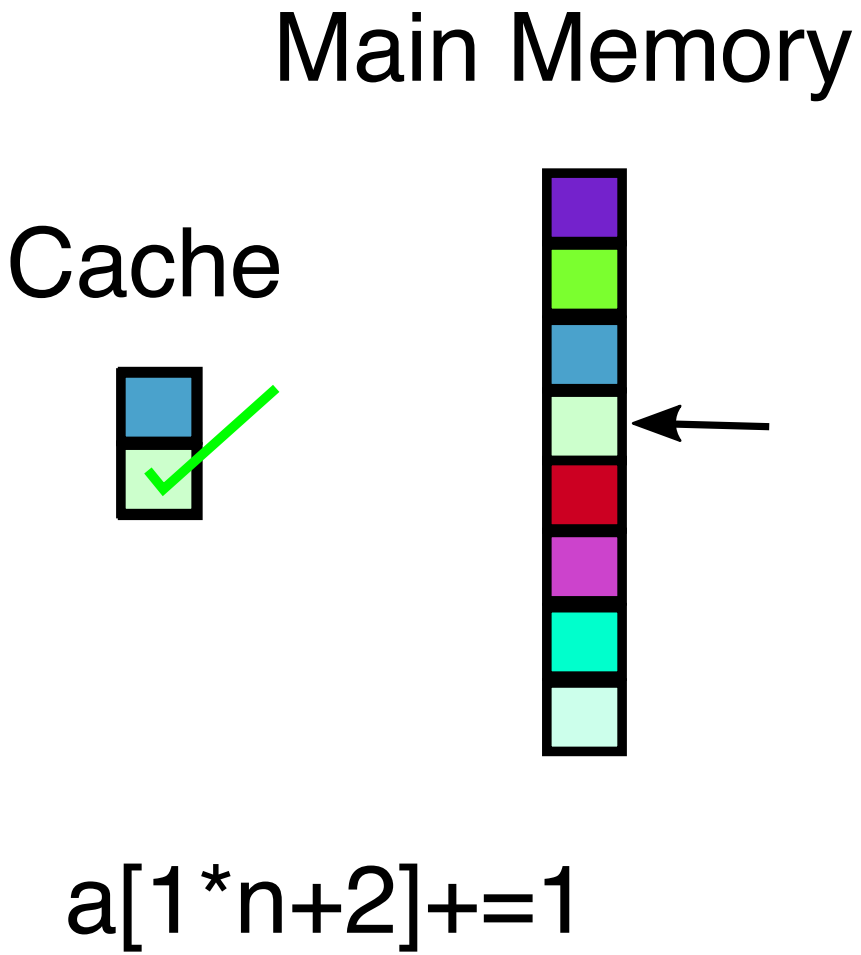
Cache: A Simple Model

next cache line is loaded...



Cache: A Simple Model

but next memory acces is a cache "hit"...



Latency Table Revisited

- Arithmetic Latency $< 1ns$
- Cache reference $< 10ns$
- Memory reference $> 100ns$

Latency Table Revisited

- Arithmetic Latency $< 1ns$
- Cache reference $< 10ns$
  - If data is in cache
- Memory reference $> 100ns$
  - If data not in cache..

How to Target Cache

- Caching is automatic
- Must understand how it works to use it
- This requires a little work, but huge payoff
- The idea is to access memory to maximize cache hits

How to Target Cache

- Access memory using "unit strides"
- We have seen a coding strategy for this already
- Blocking
  - Limiting extents of loops
  - Keeps iterates working within cache

How to Target Cache: Non-Blocked Example

```
for(int i=0;i<n;i++){
  for(int j=0;j<n;j++){
    a[i*n+j]+=1.0;
  }
}
```

How to Target Cache: Use blocking

```
for(int i=0;i<n;i+=BLOCKI){
  for(int j=0;j<n;j+=BLOCKJ){
    int iend=MIN(n,i+BLOCKI);
    int jend=MIN(n,j+BLOCKJ);
    for(int ii=i;ii<iend;ii++)
      for(int jj=j;jj<jend;jj++)
        a[ii*n+jj]+=1.0;
  }
}
```

Note: Blocking parameters deliberately left undefined here...
Rule of thumb: big enough to utilize all cache, small enough not to exceed it.

Cache Reality

- Cache is much more complicated than this
- Often multiple caches, called levels (abreviated "L")
- L1,L2,L3 on many
- Therefore you can recursively block
- each blocking level targeting a different cache

Sophisticated Blocking

```
for(int i=0;i<n;i+=BLOCKI){
  for(int j=0;j<n;j+=BLOCKJ){
    int iend=MIN(n,i+BLOCKI);
    int jend=MIN(n,j+BLOCKJ);
    for(int ii=i;ii<iend;ii+=BLOCKI2){
      for(int jj=j;jj<jend;jj+=BLOCKJ2){
        int iend2=MIN(iend,ii+BLOCKI2)
        int jend2=MIN(jend,jj+BLOCKJ2)
        for(int iii=ii;iii<iend2;iii++)
          for(int jjj=jj;jjj<jend2;jjj++)
            a[iii*n+jjj]+=1.0;
      }
    }
  }
}
```

Use more blocking levels to target more levels of cache

Sophisticated Blocking

- Can go much further
- For example: In cache mode we get a "L3" cache
- That won't fit on screen, so left as exercise!

Other Caching Issues

- Fitting in cache very significant for performance
- Cache also creates other issues
- Will briefly cover these

Cache Line Issues

- Basic unit of memory is the cache line
- Data almost always read in cache-line size chunks
  - And the cache-line size is almost always 64 bytes
- Contention arises when concurrently modifying a cache line
- This is called "false sharing"

False Sharing Example

```
float x[2]={0.0,0.0};
#pragma omp parallel num_threads(2)
{
  int id=omp_get_thread_num();
  x[id] += id;
}
```

Threads can execute in parallel, but cache lines on each thread contain identical cache lines. So, e.g. thread 0 invalidates cache of thread 1 to make update.

Cache Line Issues

- False sharing like a critical region
- Hurts performance by forcing synchronization
- Fix: keep threads one (or more) cache line away from each other
- Easily done with data padding

False Sharing Example

```
//16 floats=64 bytes
#define PAD 16
float x[2+PAD];
#pragma omp parallel num_threads(2)
{
  int id=omp_get_thread_num();
  x[id*PAD] += id;
}
```

Padding makes sure values that are accessed by different threads are not on the same cache line.

DATA ALIGNMENT Optimizations

Data Alignment

- Aligned memory can also improve cache performance
- Recall that memory is cached from cache lines
- These almost always start at an address divisible by cache line
- Easy to fix in Fortran: use the -align64byte compiler option to force all allocations to be 64 byte aligned (no code changes necessary).

Data Alignment

- Can sometimes improve performance
- Intel compiler comes with allocator(s) to make this easy
- For platform independence (at least on posix) can use posix_memalign

Data Alignment Example

```
#define CACHE_LINE (64/sizeof(float))
vector<float> x(CACHE_LINE,0.0);
for(int i=0;i<CACHE_LINE;i++){
  x[i] += 1.0;
}
```

Above code may actually need two cache lines for full operation.

Data Alignment Example

```
#define CACHE_LINE (64/sizeof(float))
vector<float> x(CACHE_LINE,0.0);
for(int i=0;i<CACHE_LINE;i++){
  x[i] += 1.0;
}
```

Although array size is one cache line, it may start *in the middle* of a cache line.

Data Alignment Example

```
#define CACHE_LINE (64/sizeof(float))
vector<float,cache_aligned_allocator<float> >
x(CACHE_LINE,0.0);
for(int i=0;i<CACHE_LINE;i++){
  x[i] += 1.0;
}
```

By using Intel's allocator here, it guarantees alignment on cache line border. Only one cache line will be used.

Data Alignment Example

```
#define CACHE_LINE (64/sizeof(float))
float* x;
posix_memalign(&x,64,CACHE_LINE*sizeof(float));
for(int i=0;i<CACHE_LINE;i++){
  x[i] += 1.0;
}
free(x);
```

Here is a more portable aligned allocation, just a little more coding required. Could easily be basis of custom allocator as well.

Data Alignment: Fortran Example

```
#define CACHE_LINE 64
double precision, allocatable :: x(:)
!DIR$ ATTIBUTES ALIGN : 64 :: x
allocate(a(CACHE_LINE))
!DIR$ ASSUME_ALIGNED : 64 :: x
do i = 1, CACHE_LINE
   x(i) = x(i) + 1.0
end do
```

Intel's compiler directives guarantee alignment on cache line border. Only one cache line will be used.

DATA LAYOUT Optimizations

Data Layout

- Another issue with memory is data layout
- Often phrased as two extremes
- Not usually a problem for Fortran: most codes use only arrays and add an extra dimension to mimic a structure.
  - Struct Of Arrays (SoA)
  - Array Of Structs (AoS)
- AoS often easier, SoA often faster

AoS: Array Of Structs

```
class ray_t{
  //Ray position.
  float x , y, z;
  //Ray momentum vector.
  float px,py,pz;
};



vector<ray> rays(nrays);
#pragma omp simd
for(auto& ray : rays){
  ray=trace(ray);
}
```

A simple raytracer in AoS style.

AoS: Array Of Structs

```
class ray_t{
  //Ray position.
  float x , y, z;
  //Ray momentum vector.
  float px,py,pz;
};



vector<ray> rays(nrays);
#pragma omp simd
for(auto& ray : rays){
  ray=trace(ray);
}
```

At best, vectorized code will be strided, but possibly also done in gather/scatter.

AoS: Array Of Structs

- Array of structs often easier
- Parallelism "free" by defining atomic function
  - the "trace" function in this case
- But it results in strided or gather/scatter vector access
- Also can cause unnecessary cache line evictions

SoA: Struct of Arrays

```
class ray_t{
  //Ray position.
  vector<float>x ,y ,z;
  //Ray momentum vector.
  vector<float>px,py,pz;
  //constructors..
};
ray_t rays(nrays);
#pragma omp simd
for(int i=0;i<nrays;i++){
  auto tmp=trace(x[i], y[i], z[i],
       px[i],py[i],pz[i]);

  //Set new x,y,z,px,py,pz
}
```

SoA: Struct of Arrays

- Struct of Arrays often faster
- But parallelism less clean
- Results in unit stride access

SoAoS

- Can also do mix between two
- Exponential number of combinations
- NP-hard optimization problem

Finding Right One

- Also changing often requires code change
- Makes changing between the two expensive

Intel SDLT

- Intel has template library called SDLT
- Automates switching between data layouts

Intel SDLT

```
class ray_t{
  //Ray position.
  float x , y, z;
  //Ray momentum vector.
  float px,py,pz;
};
SDLT_PRIMITIVE(
    ray_t,
    x,y,z,px,py,pz);
typedef sdlt::soa1d_container<ray_t> Container;

Container rays(nrays);
#pragma omp simd
for(int i=0;i<nrays;i++){
  rays[i]=trace(rays[i]);
}
```

Intel SDLT

- Wrapped object in SDLT container
- Let us keep most code unchanged
- But still can experiment with different layouts

Conclusion

- Here you learned about optimizing for memory usage
- Covered fitting in cache
- Explained alignment issues
- Finally: data layout

Practical Exercise

ARRAYS OF STRUCTURES