Performance optimisation in parallel systems

Juan Ángel Lorenzo del Castillo

CY Cergy Paris Université

ING2-GSI-MI Architecture et Programmation Parallèle

2023 - 2024

juan-angel.lorenzo-del-castillo@cyu.fr

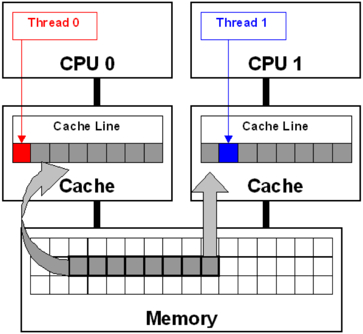
TECH

Table of Contents

TECH

Table of Contents

False sharing



Source : Intel

- It occurs when threads on different processors modify variables that reside on the same cache line. This invalidates the cache line and forces an update, which hurts performance.

False sharing

Example

```
double sum=0.0, sum_local[NUM_THREADS];

#pragma omp parallel num_threads(NUM_THREADS)
{
 int tid = omp_get_thread_num();
 sum_local[tid] = 0.0;

#pragma omp for
 for (i = 0; i < N; i++)
    sum_local[tid] += x[i] * y[i];

#pragma omp atomic
 sum += sum_local[tid];
}
```

Techniques to reduce False sharing

- Use private variables whenever it is possible.
- Modify data or iterations-to-thread affinity.
- Give more work per *chunk* to each thread.
- Use a more intelligent compiler.
- Its impact will be less noticeable on larger problems.

Table of Contents

Improving locality in distributed-memory systems

- Cannon's Matrix Multiplication Algorithm
  - Assume a matrix of size p that is a perfect square
  - Each processor gets a $n/\sqrt(p) \times n/\sqrt(p)$ chunk of data
  - Organise processors into rows and columns, keeping data locality for each processor

p(0,0)	p(0,1)	p(0,2)
p(1,0)	p(1,1)	p(1,2)
p(2,0)	p(2,1)	p(2,2)

Source : Scott B. Baden / Jim Demmel

Improving locality in distributed-memory systems

- Cannon's Matrix Multiplication Algorithm
  - Move data incrementally in $\sqrt(p)$ phases
  - Circulate each chunk of data among processors within a row or column
  - Consider iteration $i = 1, j = 2$:

```
C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]
```

A(0,0)	A(0,1)	A(0,2)
A(1,0)	A(1,1)	A(1,2)
A(2,0)	A(2,1)	A(2,2)

B(0,0)	B(0,1)	B(0,2)
B(1,0)	B(1,1)	B(1,2)
B(2,0)	B(2,1)	B(2,2)

Scott B. Baden / Jim Demmel

Improving locality in distributed-memory systems

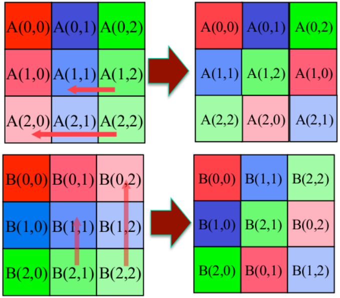
- Cannon's Matrix Multiplication Algorithm

```
C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]
```

1. Initially we want $A[1,0]$ and $B[0,2]$ to reside on the same processor as $C[1,2]$.

   So we first skew the matrices for everything to line up:

   - Shift each row *i* by *i* columns to the left.
   - Communication wraps around.
   - Same for each column.



Source : Scott B. Baden / Jim Demmel

Improving locality in distributed-memory systems

- Cannon's Matrix Multiplication Algorithm

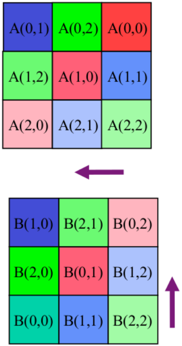
  `C[1,2] = A[1,0]*B[0,2] + ` **`A[1,1]*B[1,2]`** ` + A[1,2]*B[2,2]`

2. Then we shift rows and columns so the next pair of values $A[1,1]$ and $B[1,2]$ line up.

   We circularly shift:

   - each row by 1 column to the left.
   - each column by 1 row to the "north".

   Each processor calculates the product of the two local sub-matrices adding into the accumulated sum.



Source : Scott B. Baden / Jim Demmel

Improving locality in distributed-memory systems

- Cannon's Matrix Multiplication Algorithm

```
C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]
```

- Summary:

A(0,0)	A(0,1)	A(0,2)
A(1,0)	A(1,1)	A(1,2)
A(2,0)	A(2,1)	A(2,2)

B(0,0)	B(0,1)	B(0,2)
B(1,0)	B(1,1)	B(1,2)
B(2,0)	B(2,1)	B(2,2)

Initial A, B

A(0,0)	A(0,1)	A(0,2)
A(1,1)	A(1,2)	A(1,0)
A(2,2)	A(2,0)	A(2,1)

B(0,0)	B(1,1)	B(2,2)
B(1,0)	B(2,1)	B(0,2)
B(2,0)	B(0,1)	B(1,2)

A, B initial alignment

A(0,1)	A(0,2)	A(0,0)
A(1,2)	A(1,0)	A(1,1)
A(2,0)	A(2,1)	A(2,2)

B(1,0)	B(2,1)	B(0,2)
B(2,0)	B(0,1)	B(1,2)
B(0,0)	B(1,1)	B(2,2)

A, B after shift step 1

A(0,2)	A(0,0)	A(0,1)
A(1,0)	A(1,1)	A(1,2)
A(2,1)	A(2,2)	A(2,0)

B(2,0)	B(0,1)	B(1,2)
B(0,0)	B(1,1)	B(2,2)
B(1,0)	B(2,1)	B(0,2)

A, B after shift step 2

Source : Scott B. Baden / Jim Demmel

Improving locality in distributed-memory systems

- Limitations of Cannon's Algorithm:
  - p must be a perfect square
  - A and B must be square, and evenly divisible by $\sqrt{p}$