Intranode optimization

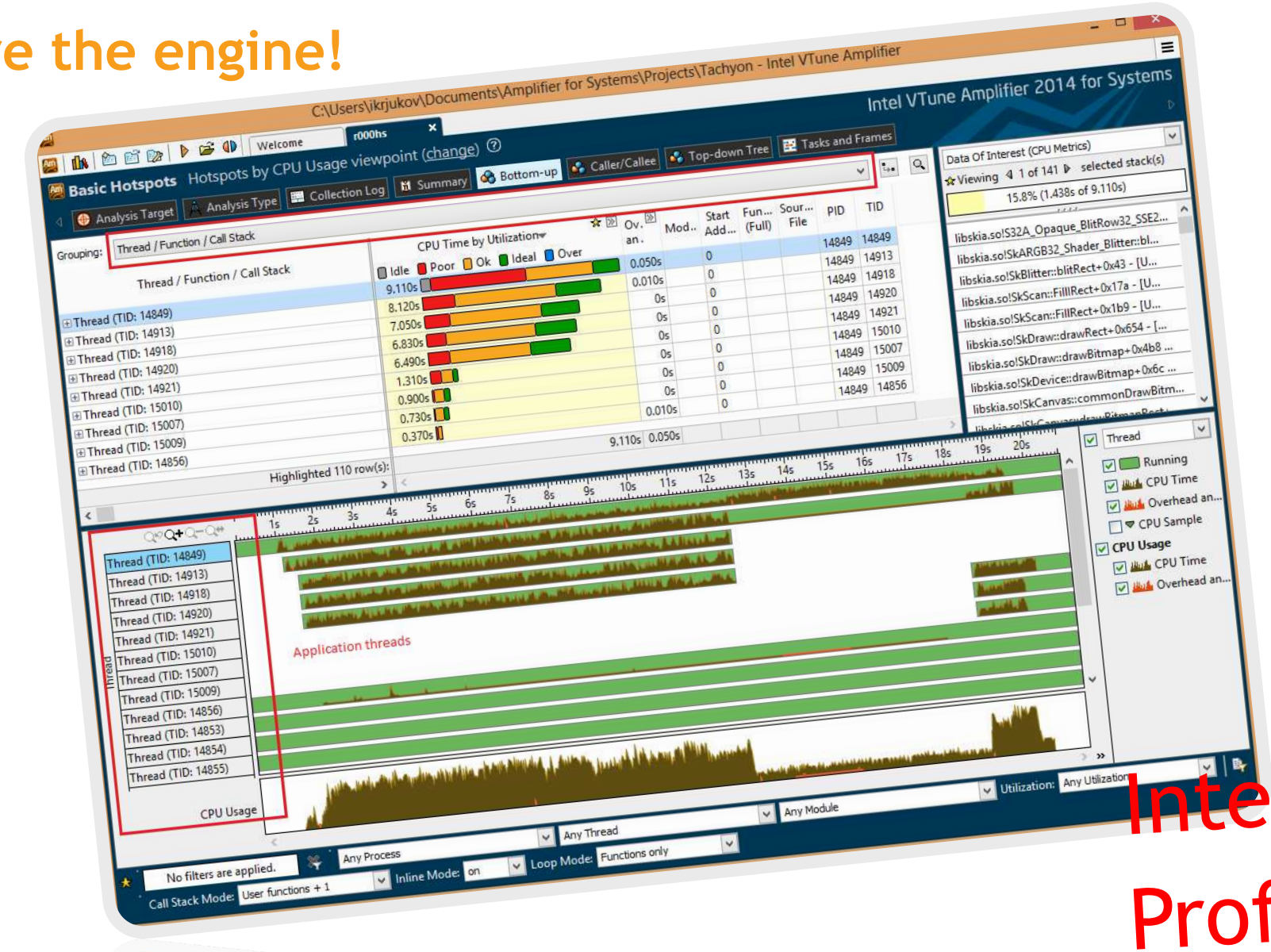
eric.pascolo@cineca.it
HPC User Support Group - CINECA

Intranode Optimization

Are you ready?

Today you become Software Mechanic!!

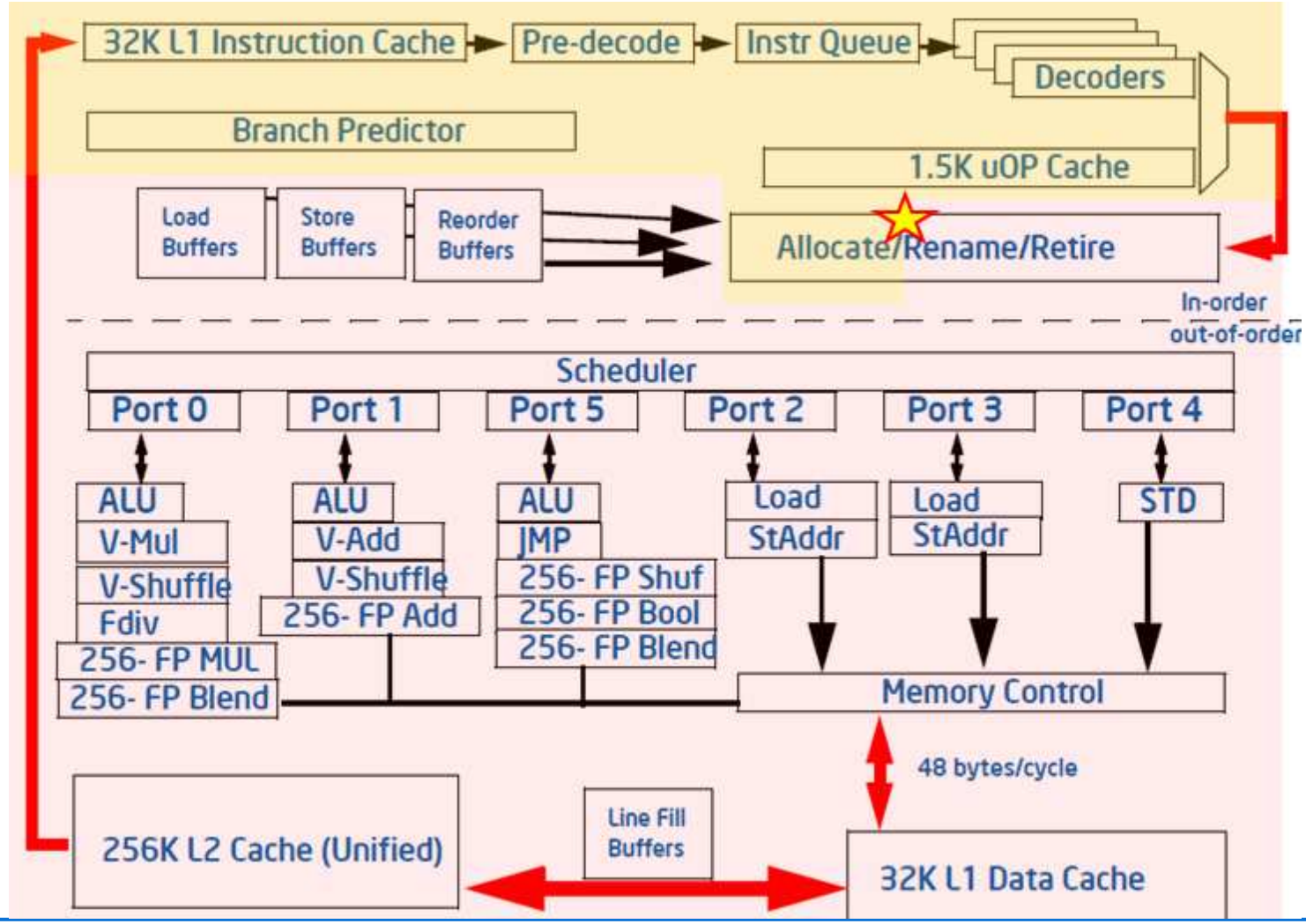
Explore the engine!



Intel Vtune Profiler

PROFILING SHOWS THE PERFORMANCE OF HARDWARE AND SOFTWARE COUPLED

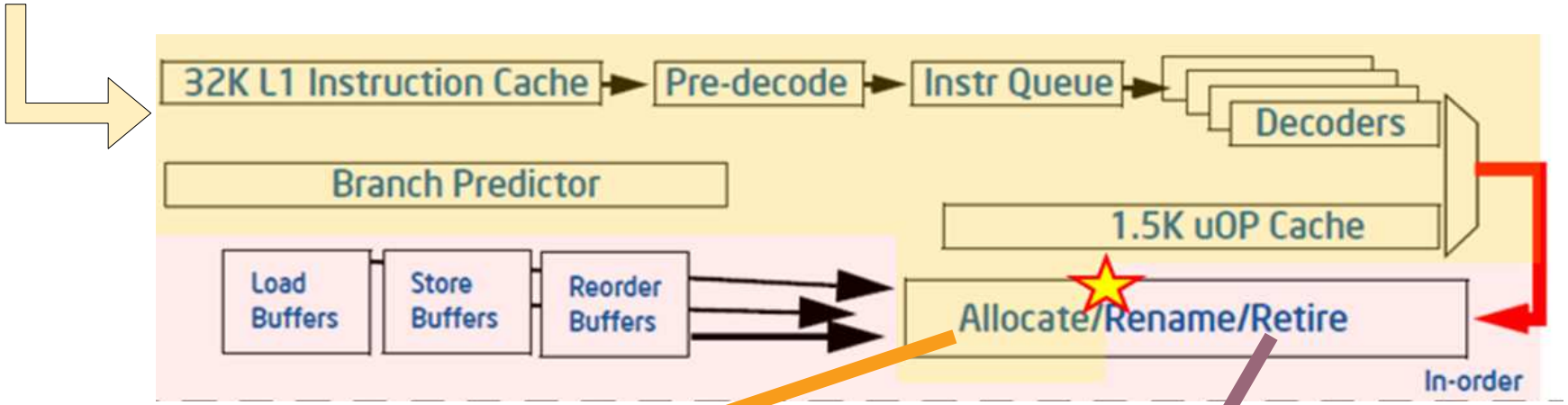
Modern Processor Pipeline



Front End

Compiled CODE

Ideal : all uOP are retired

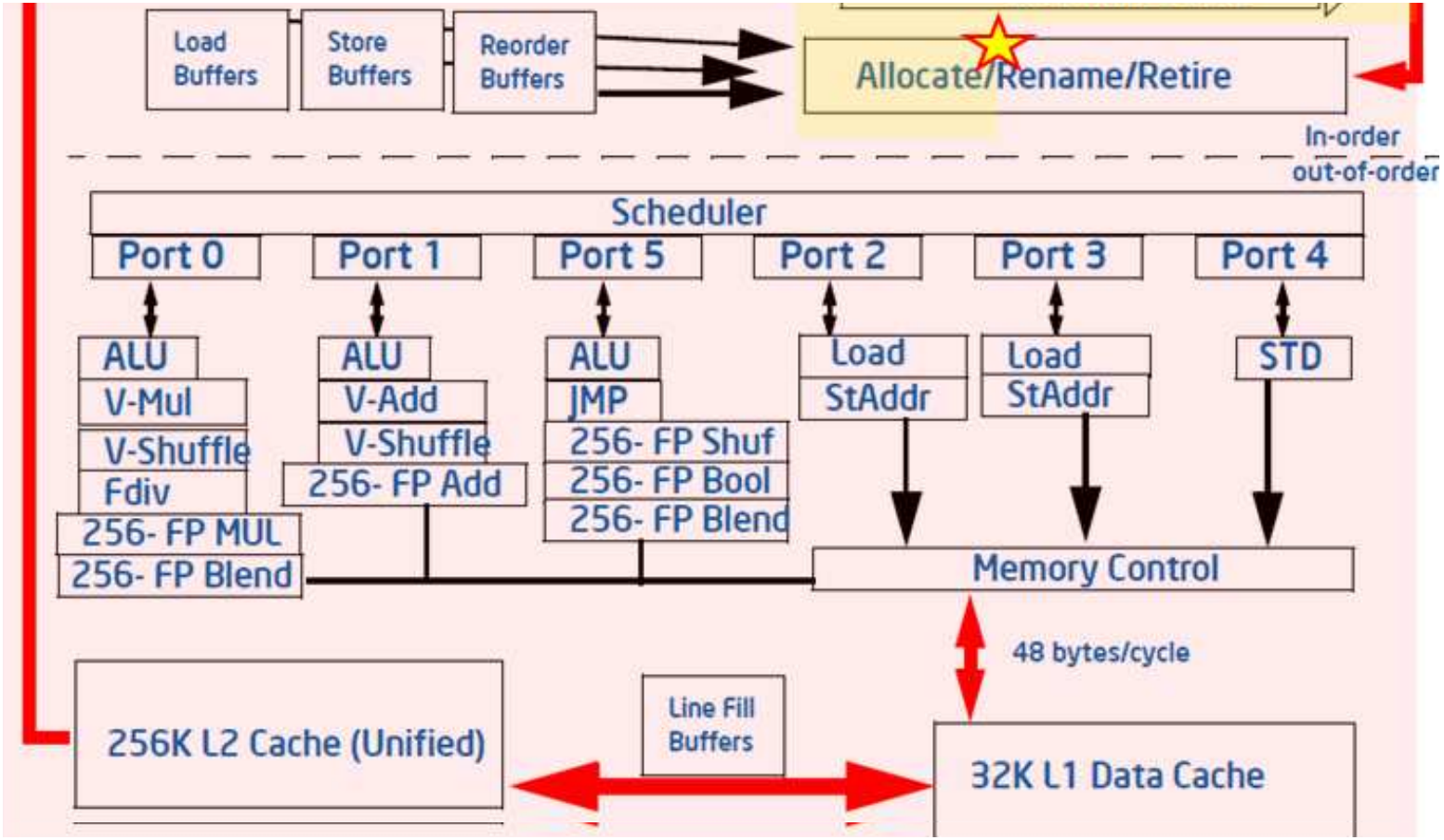


Allocation:
process that
fed the BE of uOP

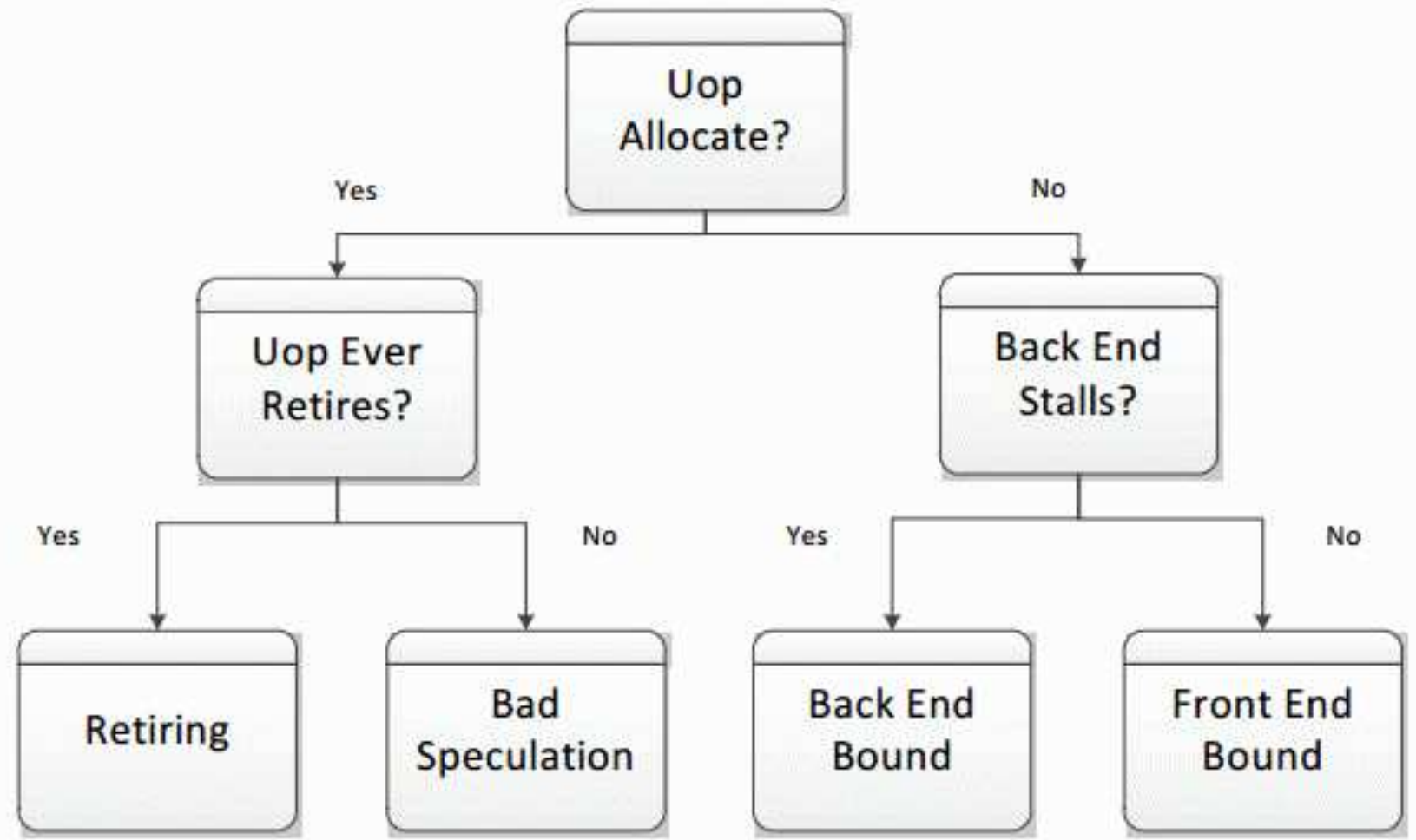
Retirement:
The completion of a uOp's execution. The results are committed to the architectural state.

4 uOP/cycle(pipeline slot) => *CPI* = 0.25

Back End


Load Buffers
Store Buffers
Reorder Buffers
Allocate/Rename/Retire
In-order
out-of-order
Scheduler
Port 0
Port 1
Port 5
Port 2
Port 3
Port 4
ALU
V-Mul
V-Shuffle
Fdiv
256- FP MUL
256- FP Blend
ALU
V-Add
V-Shuffle
256- FP Add
ALU
JMP
256- FP Shuf
256- FP Bool
256- FP Blend
Load
StAddr
Load
StAddr
STD
Memory Control
48 bytes/cycle
256K L2 Cache (Unified)
Line Fill Buffers
32K L1 Data Cache

Intel's Diagram



FE,Retiring and Bad Speculation

Retiring : the performance issues are probably due to an heavy use of micro sequencer (assistant generator of long stream of uOPs)

Bad Speculation : when the pipeline is busy fetching and executing non-useful operations due to incorrect speculation.

Front End: problem related to code layout, try to solve using PGO(Profiling guided optimization).

PROFILE GUIDED OPTIMIZATION

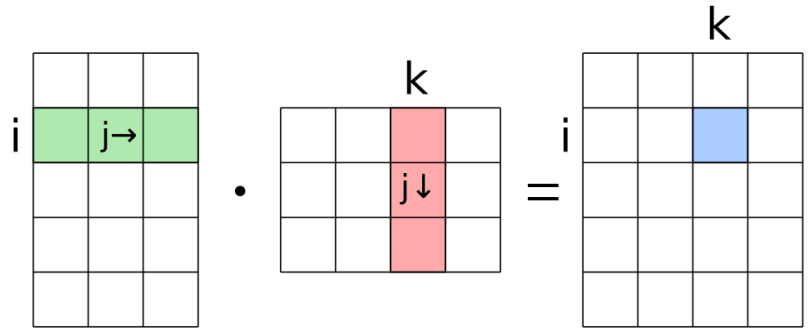
icc **-prof_gen** code.c

icc **-prof_use** code.c

Launch code.xx
On typical dataset

What is the difference?

Matrix Mul

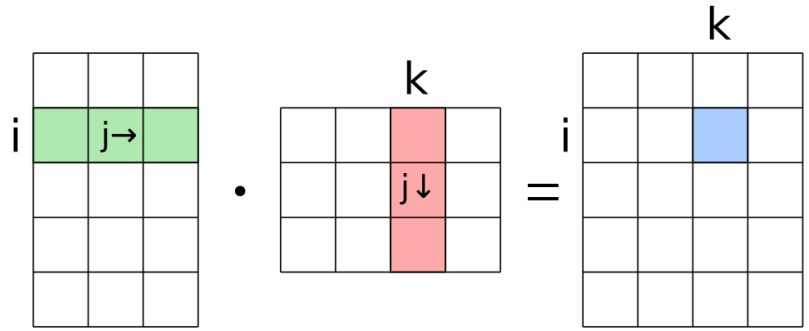


AXPI

a*X+Y

What is the difference?

Matrix Mul



CORE BOUND

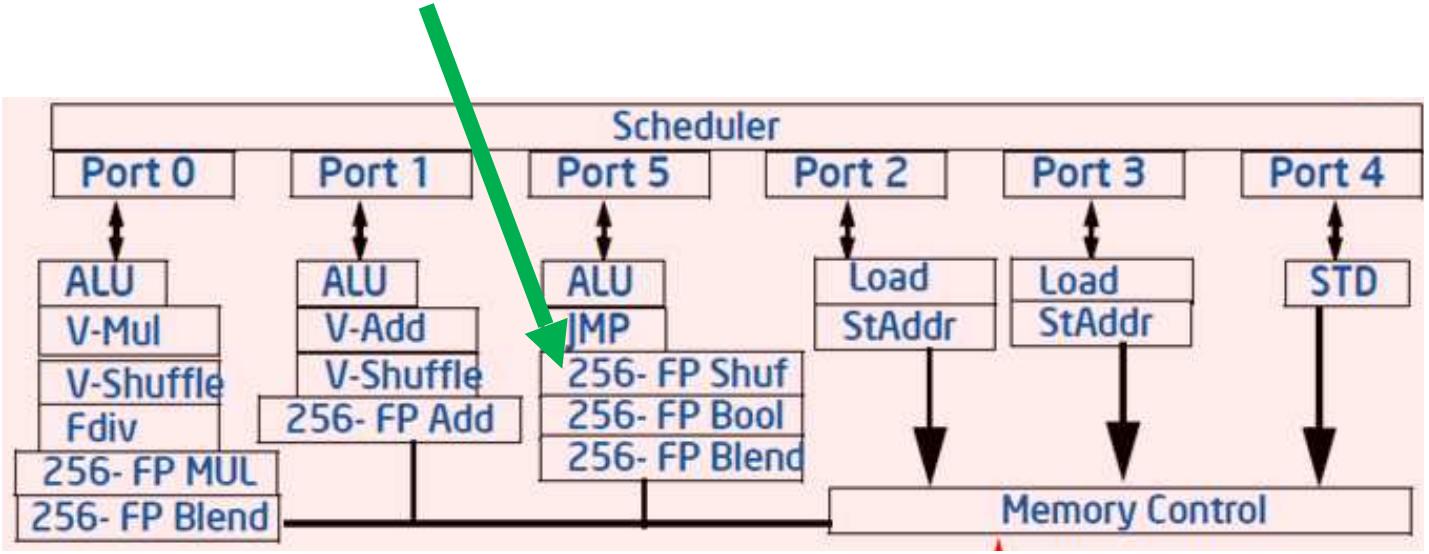
AXPI

a*X+Y

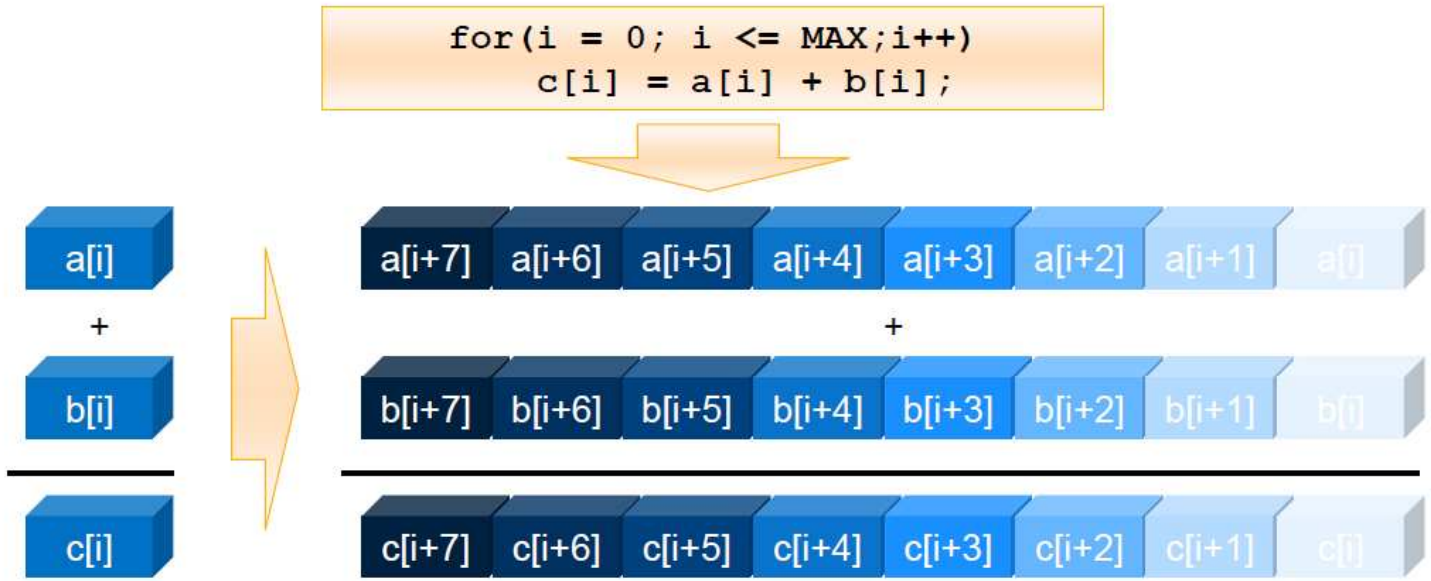
MEM BOUND

Increase Performance

256 Vector Register



Single Instruction Multiple Data



Many Ways

Compiler Auto Vectorization

Compiler Assist Vectorization

Intrinsic Function

Assembler code

Compiler AutoVectorization

icc -x<ARCH>

icc -ax<ARCH1>,<ARCH2>

icc -no-vec

CINECA MARCONI ARCH = avx2

To see code optimization report
icc -qopt-report 5

Reason for Vectorization Fails

Data dependencies

Wrong Alignement

Non-unit stride access

Non Vec Math functions

Function calls

Loop body Too complex

Data Alignment

Data alignment means putting the data at a memory address equal to some multiple of the word size (AVX 256 bit).

```
void * _mm_malloc(int size, int word)
```

Remember to compiler that an array is aligned

```
__assume_aligned(var,word);
```

Data Dependency

Flow dependence

```
for(int i;..){
        X[i] =
        … = X[i]
}
```

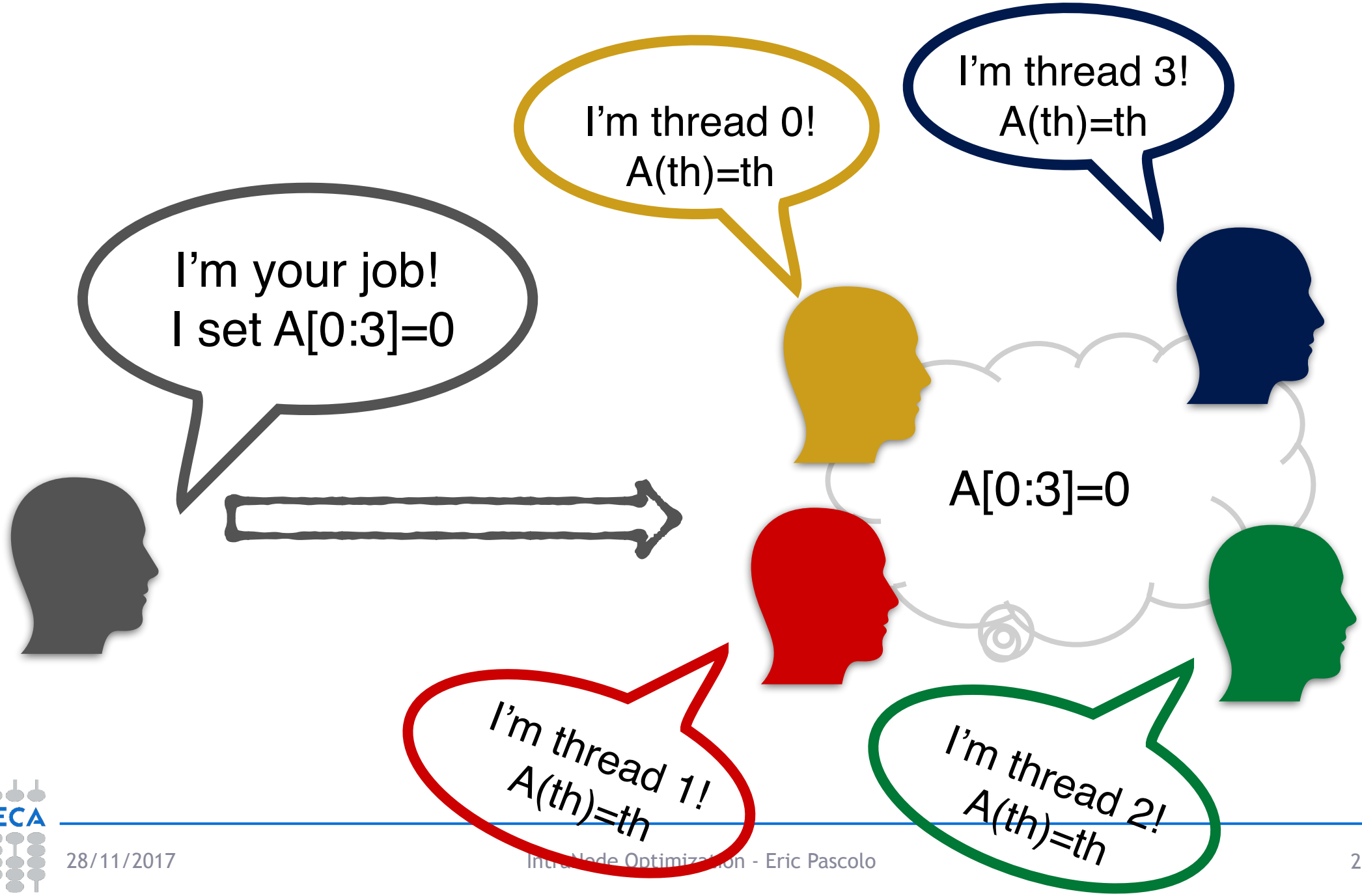
Anti dependence

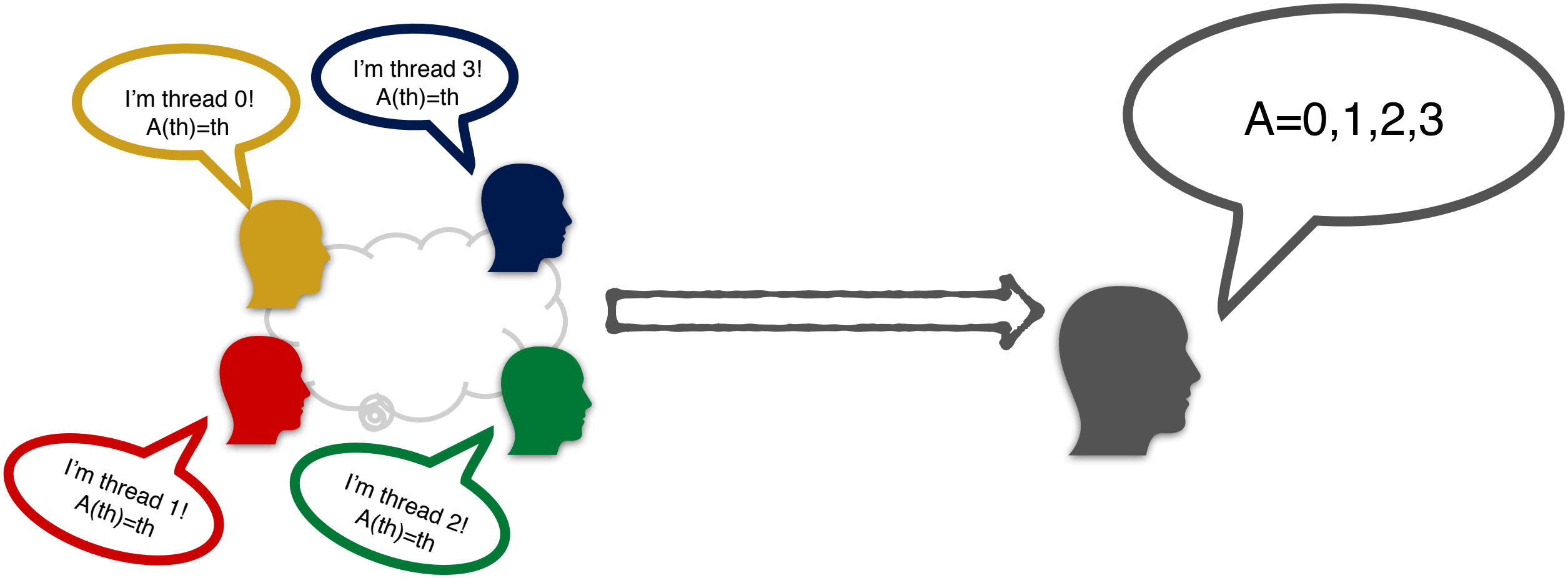
```
for(int i;..){
        … = X[i]
        X[i] = …
}
```

Output dependence

```
for(int i;..){
        X[i] = …
        X[i] = …
}
```

- OpenMP is multithreads library
- Fork and Join schema
- Version 4.5
- Not change code, add #pragma


I'm your job!
I set A[0:3]=0
I'm thread 0!
A(th)=th
I'm thread 3!
A(th)=th
A[0:3]=0
I'm thread 1!
A(th)=th
I'm thread 2!
A(th)=th


I'm thread 0!
A(th)=th
I'm thread 3!
A(th)=th
I'm thread 1!
A(th)=th
I'm thread 2!
A(th)=th
A=0,1,2,3

OpenMP Cluses

- Private
- Firstprivate
- Shared
- default
- Lastprivate

Example

```
int main(int, char **)
{
int num_threads,myid;
myid = 0;
num_threads = 1;
 #pragma omp parallel private(myid,num_threads)
{
        num_threads = omp_get_num_threads();
        myid = omp_get_thread_num();
        cout<<"Hello world! I'm "<<myid<<" of <<num_threads<<"\n";
}}
```

```
int main(int, char **)
{
int num_threads,myid;
myid = 0;
num_threads = 1;
 #pragma omp parallel private(myid,num_threads)
{
        num_threads = omp_get_num_threads();
        myid = omp_get_thread_num();
        #pragma omp critical
        cout<<"Hello world! I'm "<<myid<<" of <<num_threads<<"\n";
}}
```

Factorial

```
#include <omp.h>
int main ()
{       double var = 10000;
        omp_set_num_threads(4);
        #pragma omp parallel for
                for( int i =var-1; 1<i ; i-- )
                        var = i * var ;
        printf( "Factorial =  %d" , var );   }
```

Is it ok?!

```
#include <omp.h>
int main ()
{       double var = 10000;
        omp_set_num_threads(4);
        #pragma omp parallel for reduction(*:var)
                for( int i =var-1; 1<i ; i-- )
                        var = i * var ;
        printf( "Fattoriale =  %d" , var );   }
```

ImageEngine v 0.1