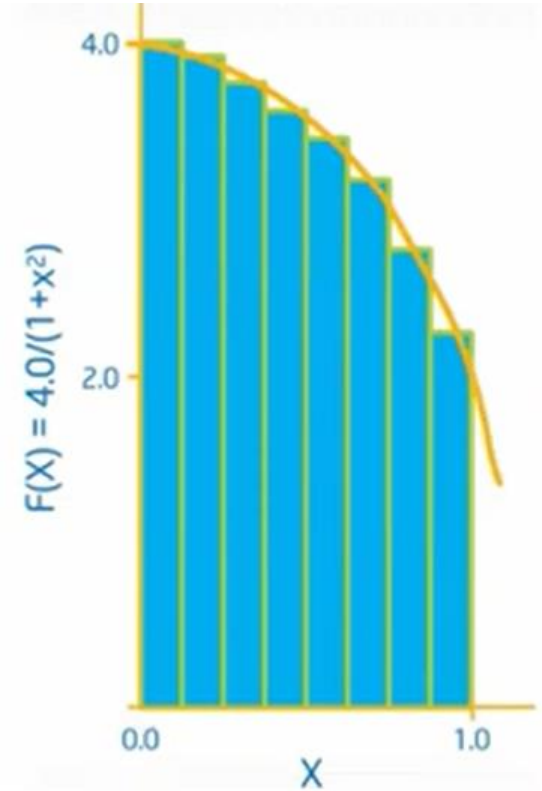
Universidade de Brasília
Faculdade do Gama
Engenharia de Software

PSPD – Programação OMP

Fernando W Cruz

Comparando soluções OMP – medindo nível de paralelização

Cálculo de PI



Matematicamente, sabemos que:

$$\int_0^1 \frac{4.0}{1+x^2}\,dx = \pi$$

Podemos aproximar essa integral como a soma de retângulos:

$$\sum_{i=0}^{n} F(x_i)\Delta x \cong \pi$$

Onde cada retângulo tem largura Δx e altura $F(x_i)$ no meio do intervalo i.

Exemplo retirado dos slides do Prof. Marco Zanata (UFPR)
https://www.youtube.com/watch?v=2F_GpRQ2mvI

Cálculo de PI

```
static long num_steps = 100000;
double step;
int main () {
  int i; double x, pi, sum = 0.0;
  step = 1.0/(double) num_steps;
  for (i=0;i< num_steps; i++){
    x = (i + 0.5) * step; // Largura do retângulo
    sum = sum + 4.0 / (1.0 + x*x); // Sum += Área do retângulo
  }
  pi = step * sum;
}
```

Cálculo de PI

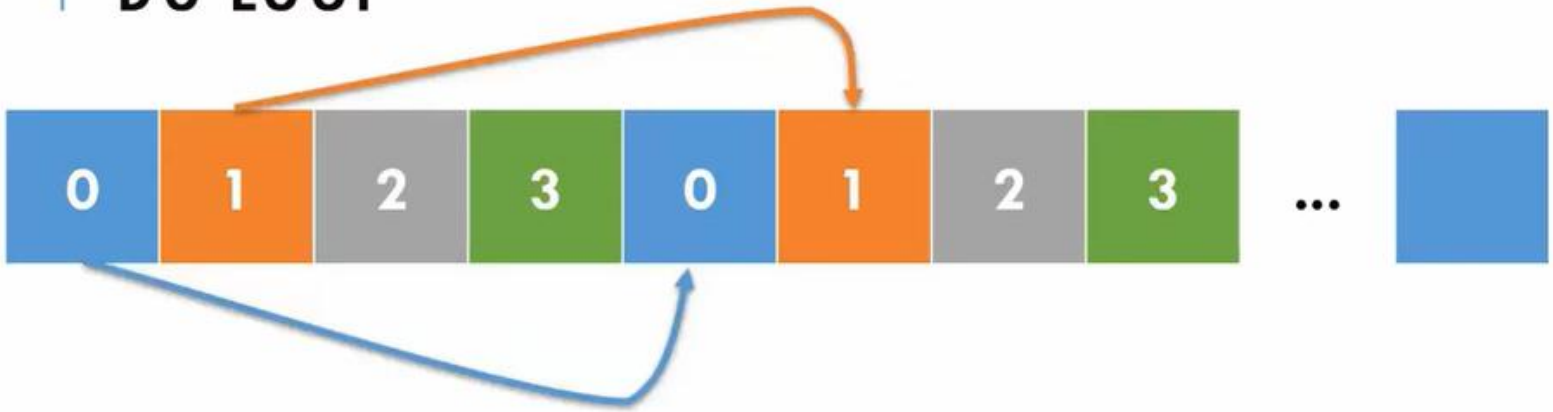
```
#include <omp.h>
static long num_steps = 100000; double step;
#define NUM_THREADS 2
void main () {
  int i, nthreads; double pi, sum[NUM_THREADS];
  step = 1.0/(double) num_steps;
  #pragma omp parallel num_threads(NUM_THREADS)
  {
    int i, id,nthrds; double x;
    id = omp_get_thread_num();
    nthrds = omp_get_num_threads();
    if (id == 0) nthreads = nthrds;
    for (i=id, sum[id]=0.0; i<num_steps; i=i+nthrds) {
      x = (i+0.5)*step;
      sum[id] += 4.0/(1.0+x*x);
    }
 }
 for(i=0, pi=0.0; i<nthreads; i++)
   pi += sum[i] * step;
}
```

Promovemos um escalar para um vetor dimensionado pelo número de threads para prevenir condições de corrida.

Usamos uma variável global para evitar perder dados

Cálculo de PI

DISTRIBUIÇÃO CÍCLICA DE ITERAÇÕES DO LOOP

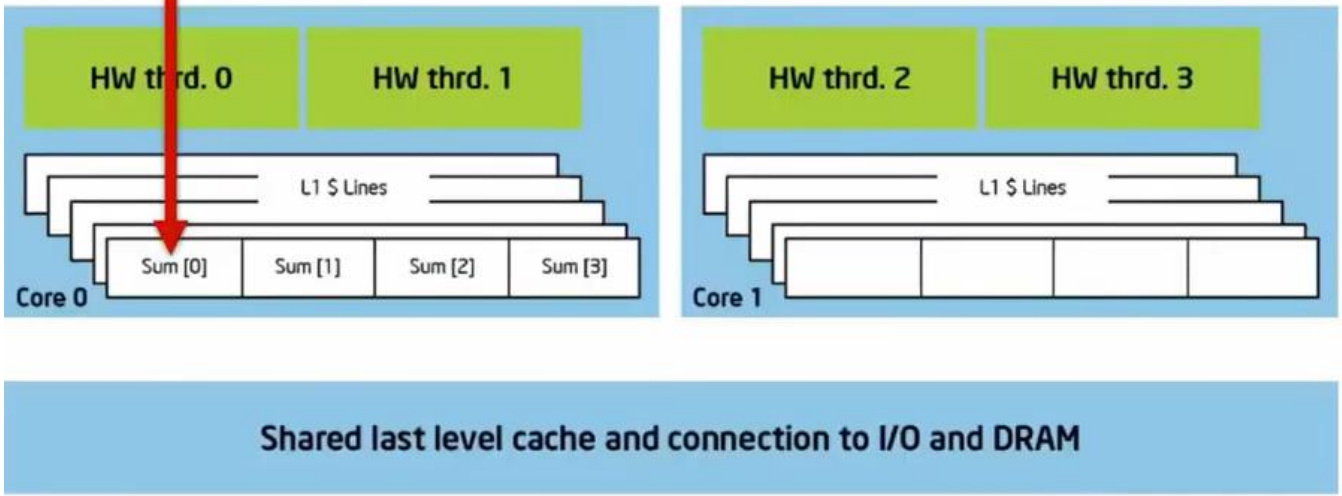



```
// Distribuição cíclica
for(i=id;  i<num_steps;  i += nthreads;)
```

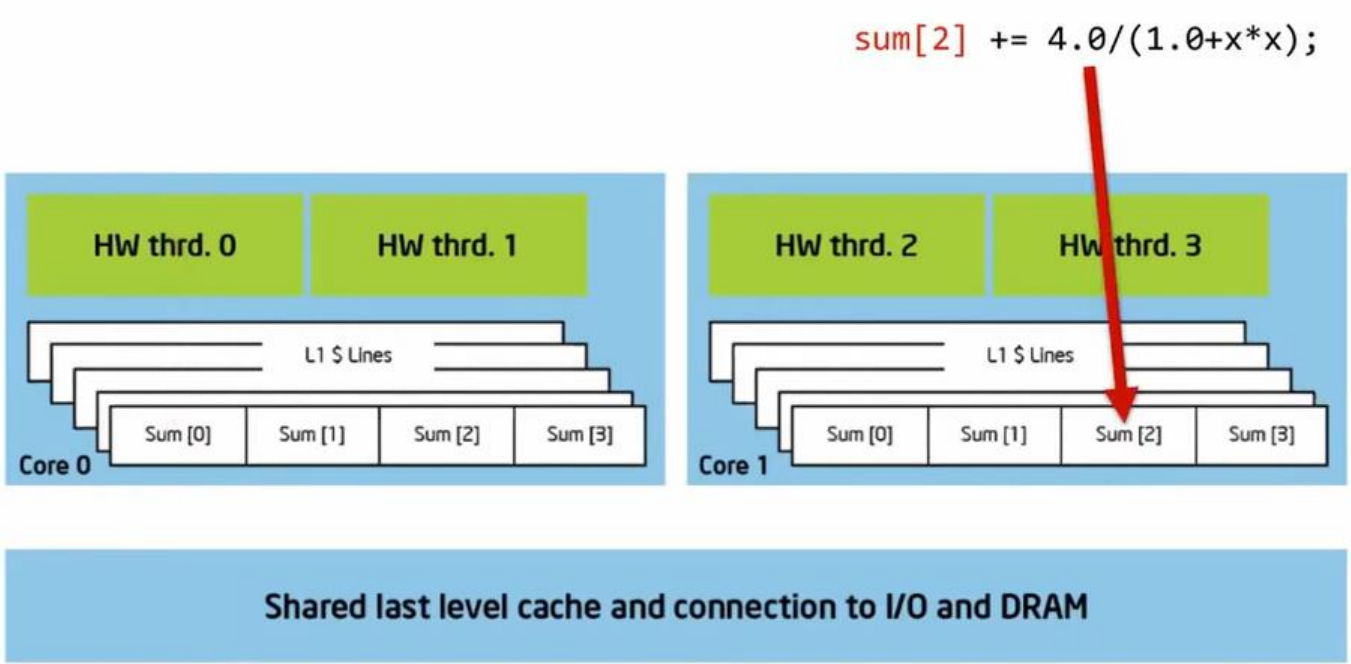
Cálculo de PI

O MOTIVO DESSA FALTA DE DESEMPENHO? FALSO COMPARTILHAMENTO

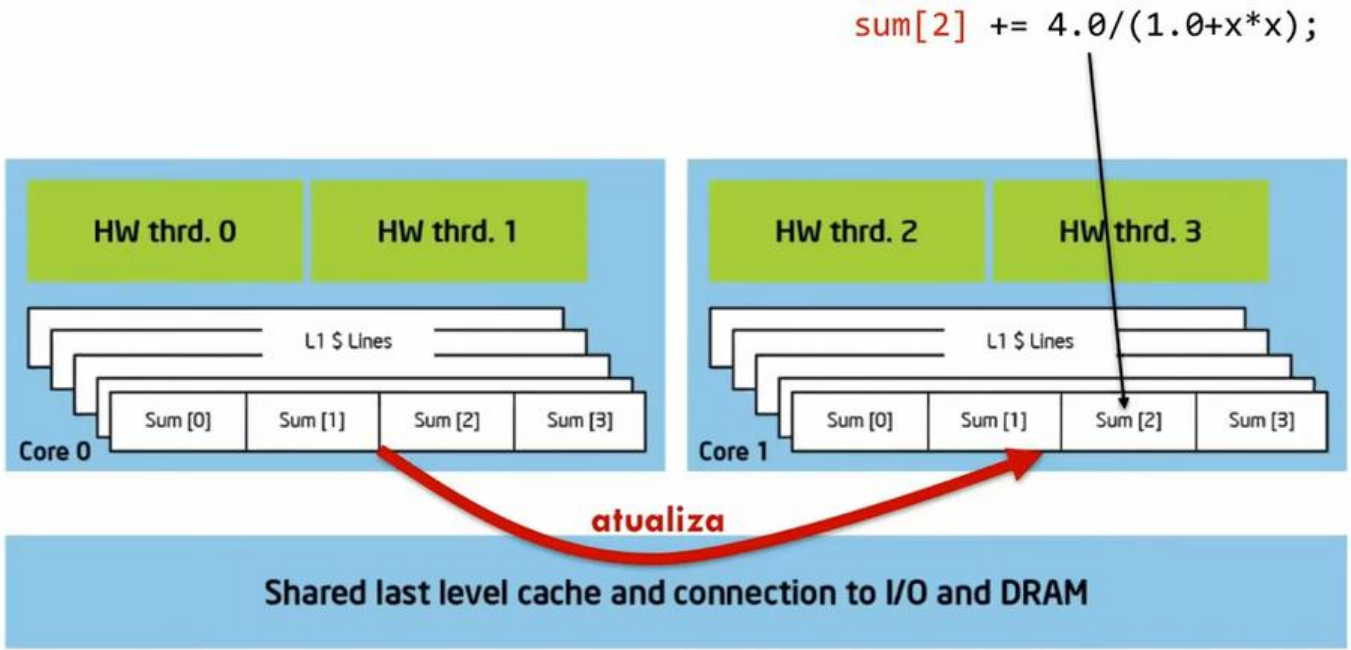
```
sum[0] += 4.0/(1.0+x*x);
```



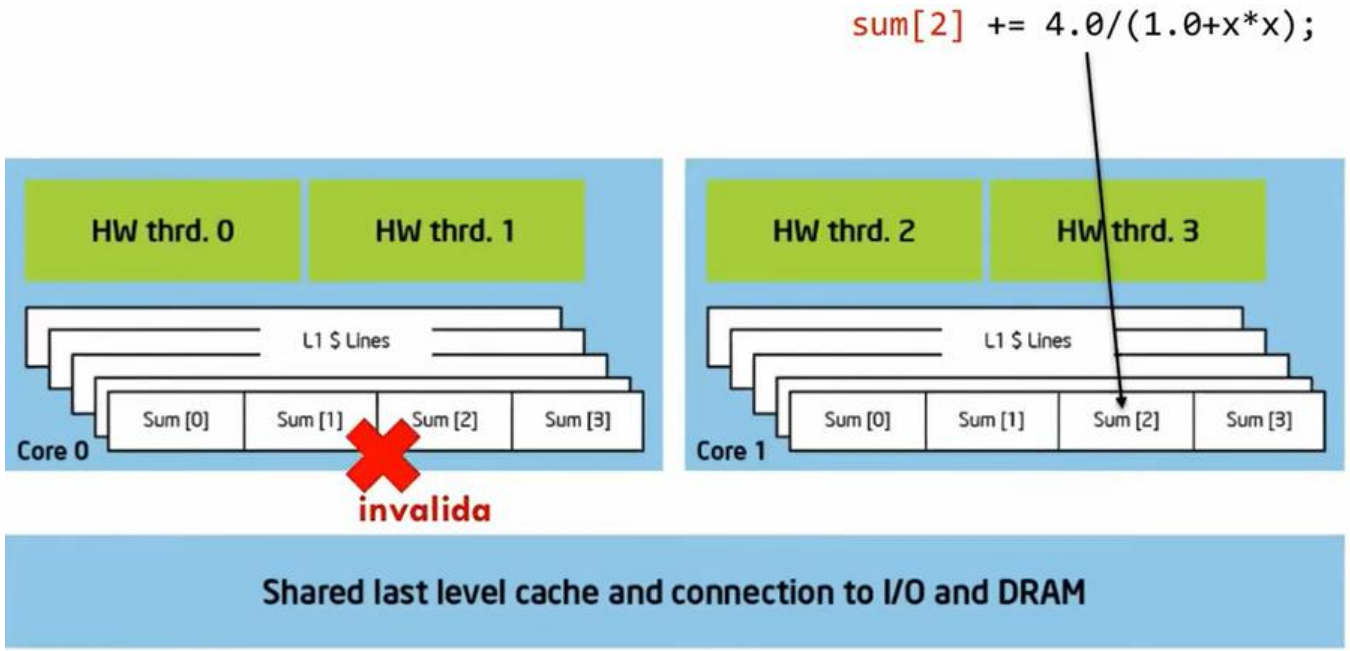
Cálculo de PI



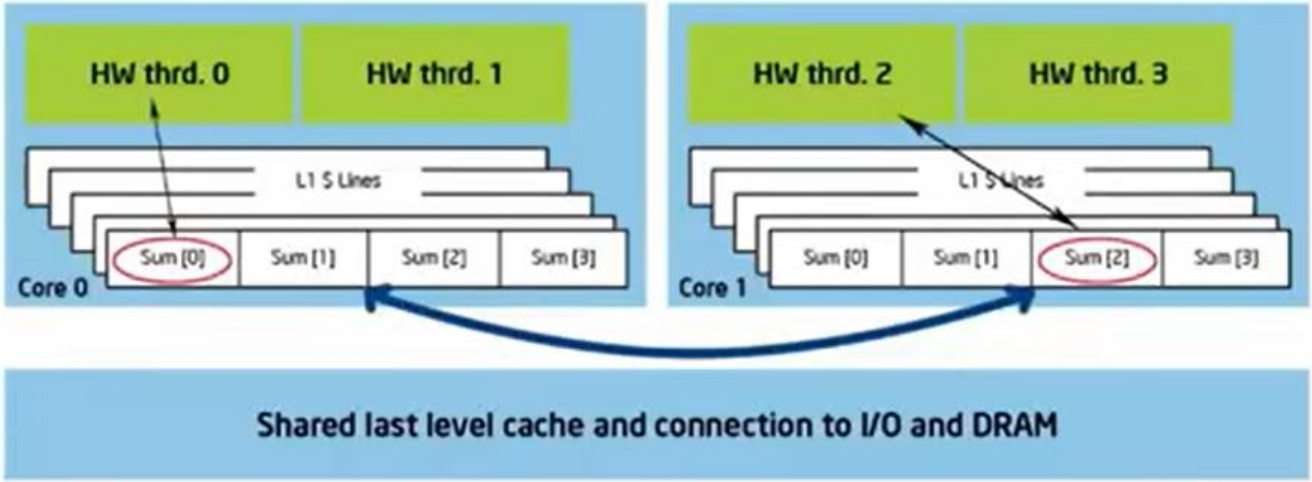
Cálculo de PI



Cálculo de PI



Cálculo de PI

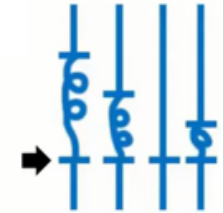


Se promovermos escalares para vetores para suportar programas SPMD, os elementos do vetor serão contíguos na memória, compartilhando a mesma linha de cache... Resultando em uma baixa escalabilidade.

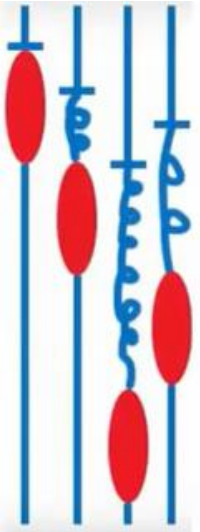
Solução: Colocar espaçadores "Pad" para que os elementos usem linhas distintas de cache.

Sincronização – alternativas

- Assegura que uma ou mais threads estão em um estado bem definido em um ponto conhecido da execução
- As duas formas de sincronização são:
  - Barreira: cada thread espera na barreira até a chegada das demais



  - Exclusão mútua: define um bloco de código onde apenas uma thread pode executar por vez



Universidade de Brasília
Faculdade do Gama
Engenharia de Software

PSPD – OMP

Fernando W Cruz