SFU SIMON FRASER UNIVERSITY
THINKING OF THE WORLD

MIAL
MEDICAL IMAGE ANALYSIS LAB

An Evaluation of Parallelization Techniques for MRF Image Segmentation

Shane Mottishaw, Sergey Zhuravlev, Lisa Tang, Alexandra Fedorova, and Ghassan Hamarneh

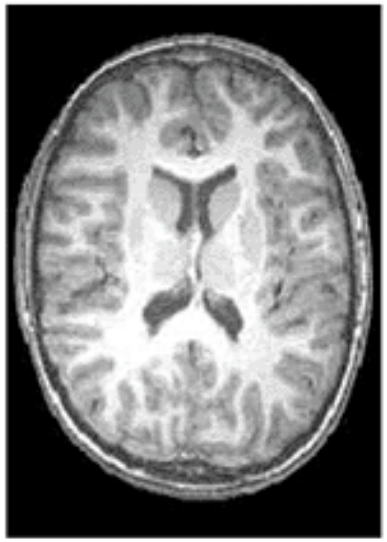
Presnted by: Ahmed Saad

Medical Image Analysis Lab,
School of Computing Science, Simon Fraser University, Canada

smottish@sfu.ca

Motivation

- MRFs are of great interest to the medical image community
  - Captures spatial information
  - Many successful applications
- Problems:
  - Parameter selection
  - Computational complexity



smottish@sfu.ca

Motivation

- Solution: acceleration via parallelization
  - Mitigates computational complexity
  - Enables real-time interaction

smottish@sfu.ca

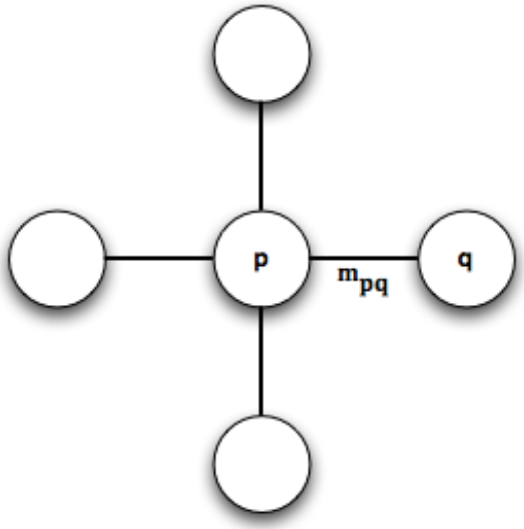
Evaluation

- Chose Belief Propagation (BP) MRF optimization
- Parallelize for multi-core processors
- Why not GPU?
  - Lack of global synchronization
  - Programming model can reduce productivity and performance

smottish@sfu.ca

BP Overview

- Image represented as a graph
  - Nodes represent pixels
  - Edges defined by neighbourhood
- Nodes pass messages to their neighbours
  - Node p passes message m_{pq} to node q
- After T iterations, accumulated messages determine labeling



smottish@sfu.ca

BP Messages

- In iteration t, node p sends message m_{pq}^{t} to node q:

$$m_{pq}^{t}(f_q) = \min_{f_p}\left(V(f_p, f_q) + D_p(f_p) + \sum_{s \in N(p)\setminus q} m_{sp}^{t-1}(f_p) \right)$$

smoothness term

data term

received messages from t-1

- m_{pq}^{t} is a vector with size = number of labels

smottish@sfu.ca

BP Labelling

- After T iterations, a pixel's label b_q is calculated as:

$$b_q(f_q^*) = D_q(f_q^*) + \sum_{p \in N(q)} m_{pq}^t(f_q^*)$$

- Where f_q^* is the optimal label

smottish@sfu.ca

Parallelizing BP: Dependencies

- If two sections/blocks of code share heap data, **data dependency** exists
- In BP, neighbours will read/write shared memory to send messages
- There is also a dependency between iterations

Thread 0

```
msg[q][p] = calc(msg[p][q])
```

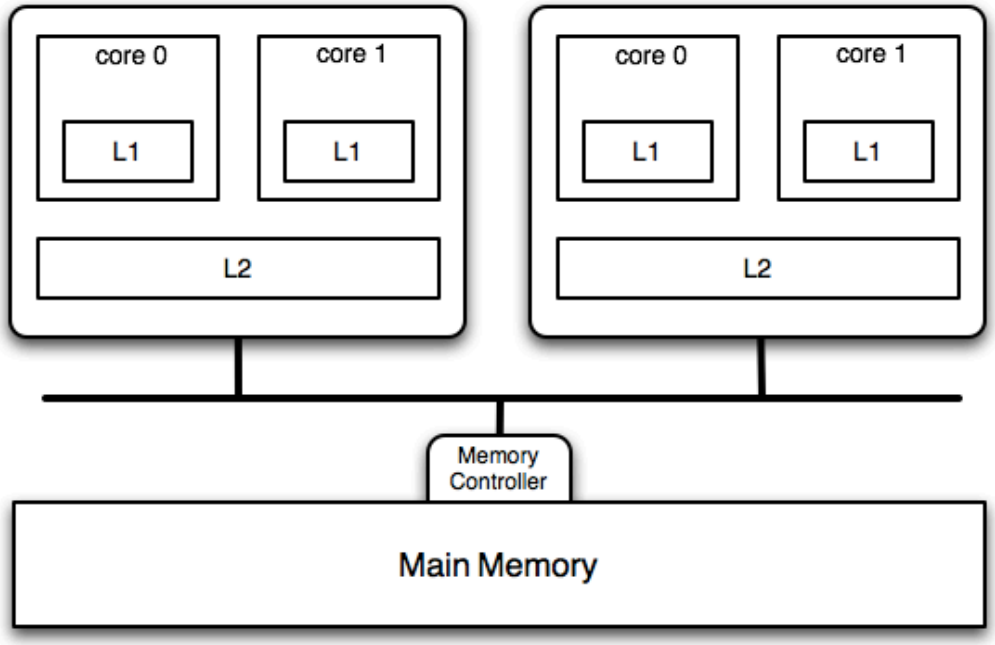
Thread 1

```
msg[p][q] = calc(msg[q][p])
```

smottish@sfu.ca

Parallelizing BP: Inter-processor communication

- Concurrent shared memory access means:
  - Inter-processor/inter-thread communication, and/or
  - Synchronization
- Communication increases latencies and contention

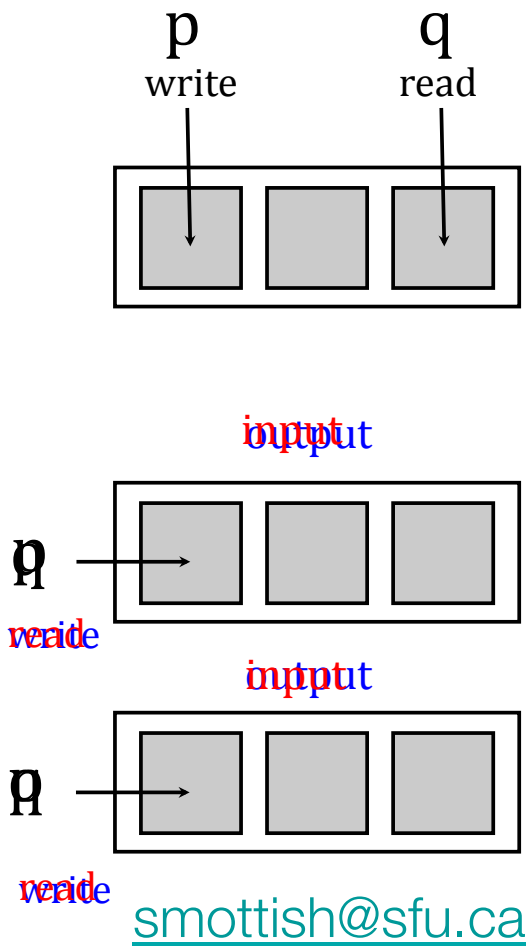


- Must carefully manage data and synchronization!

smottish@sfu.ca

Implementation

- "Typical" implementation
  - Neighbours share a queue to send/receive messages
  - Problem: requires synchronization for each read/write!
- Double buffering
  - Separate read/write or input/output queues
  - Single synchronization step to "swap"



smottish@sfu.ca

Implementation

- In iteration t, synchronization required to guarantee that:
  - A node has received all its messages from t-1
  - A node only sends to neighbour if neighbour also in t

smottish@sfu.ca

Implementation: Partitioning

- Each thread works on an equal partition of nodes
- Each thread is bound to a core
- When message sent from node in partition X to node in partition Y, inter-thread/inter-core communication
- Assign “neighbouring partitions” to threads/cores on same chip

smottish@sfu.ca

Implementation: Message Passing

Algorithm 1: parallel message-passing routine executed by each thread

```
   Input: number of iterations T and a set of nodes N
 1 begin
 2     for t ← 1 to T do
 3         foreach p ∈ N do
 4             foreach q ∈ neighbours(p) do
 5                 message ← m^t_pq
 6                 while stateOf(q) ≠ RESET do
                       ▷loop until condition met i.e. spin
 7                 end
 8                 sendMsg(message)
 9             end
10             while stateOf(p) ≠ FULL do
                   ▷loop until condition met i.e. spin
11             end
12             swapBuffers(p)
13             stateOf(p) ← RESET
14         end
15     end
16 end
```

smottish@sfu.ca

Implementation: “lock-step”

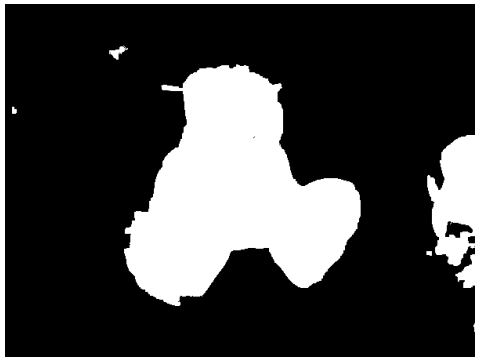
Algorithm 2: parallel lock-step routine executed by each thread

```
   Input: number of iterations T and a set of nodes N
 1 begin
 2     for t ← 1 to T do
 3         foreach p ∈ N do
 4             foreach q ∈ neighbours(p) do
 5                 message ← m_pq^t
 6                 sendMsg(message)
 7             end
 8         end
 9         spinWait()
10         swapBuffers(p)
11         spinWait()
12     end
13 end
```

smottish@sfu.ca

Experimental Setup

- Binary image segmentation
  - 600x450 greyscale image
  - 15 iterations

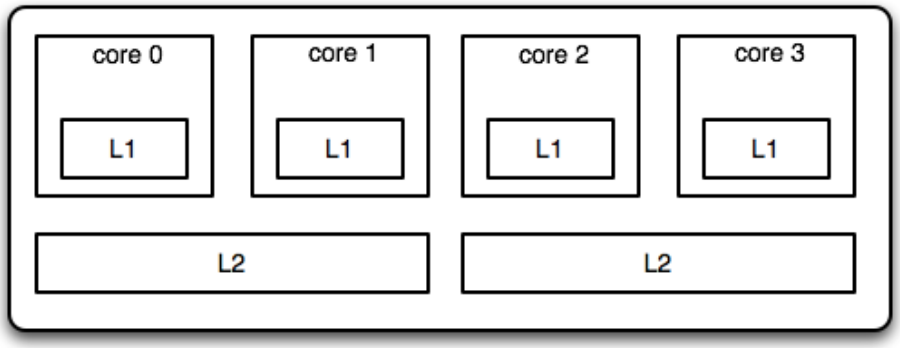


- Measure the time it takes for all threads to complete an iteration
- Run-time = sum of iteration times
- Averaged over 10 trials

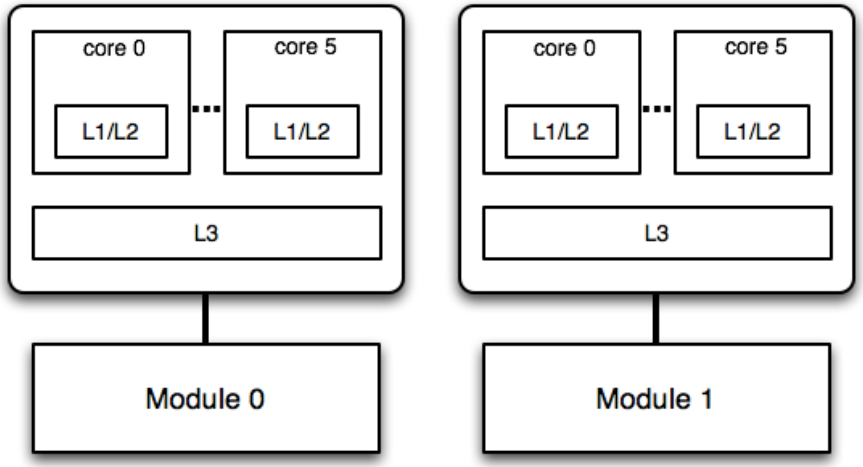
smottish@sfu.ca

Experimental Setup: Machines

2x 4-core Intel Xeon E5405



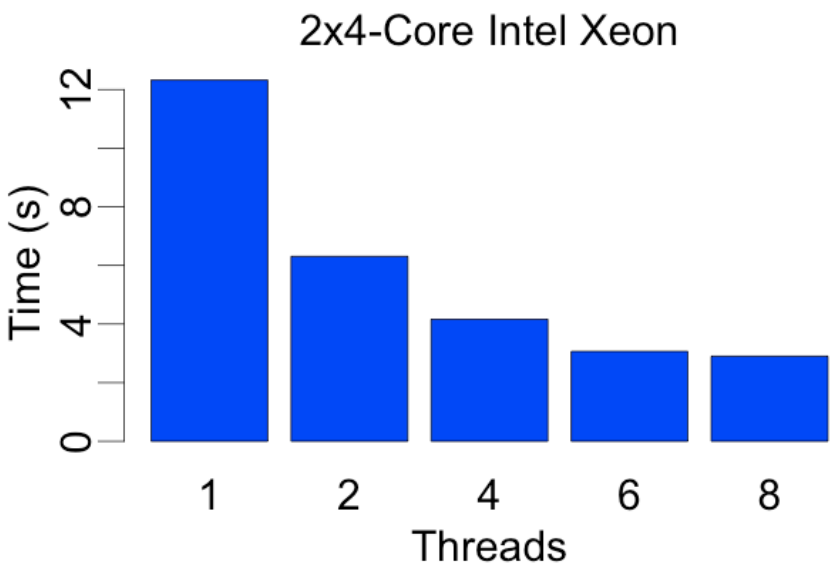
4x 6-core AMD Opteron 2435

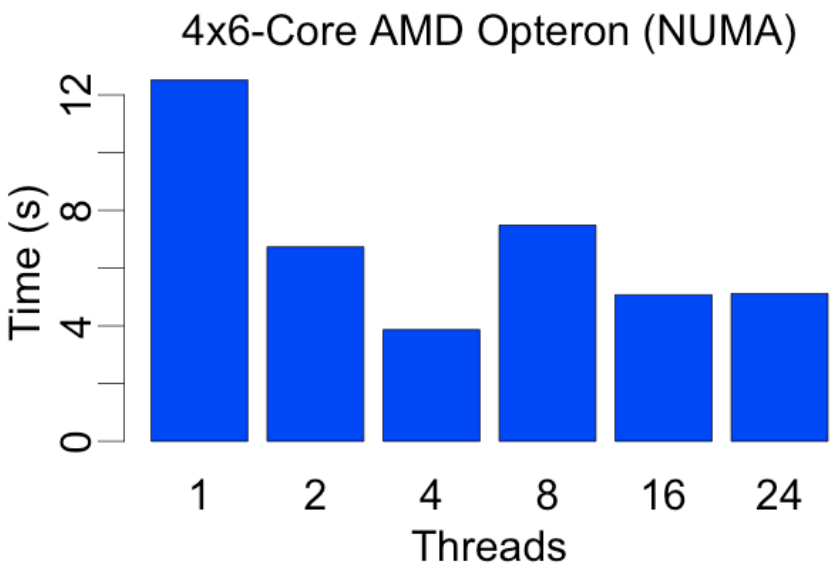


smottish@sfu.ca

Results: “Message Passing”

- Run-times

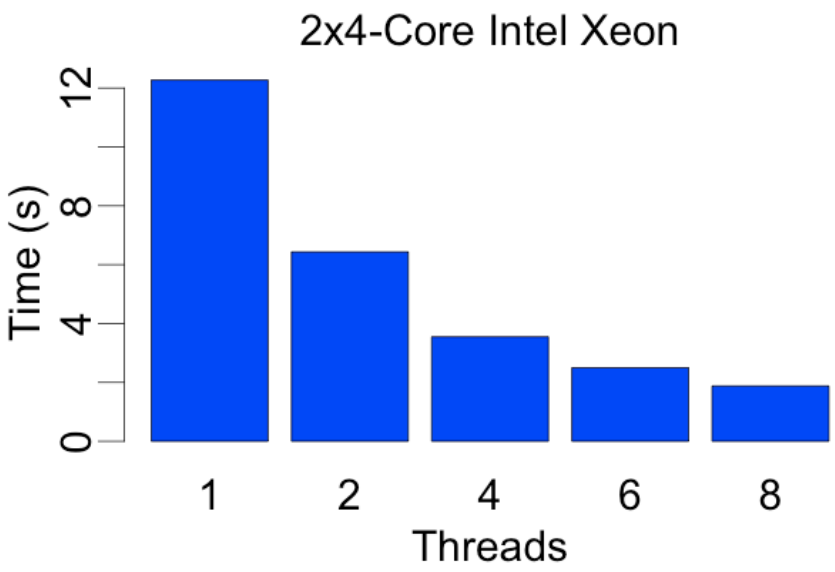


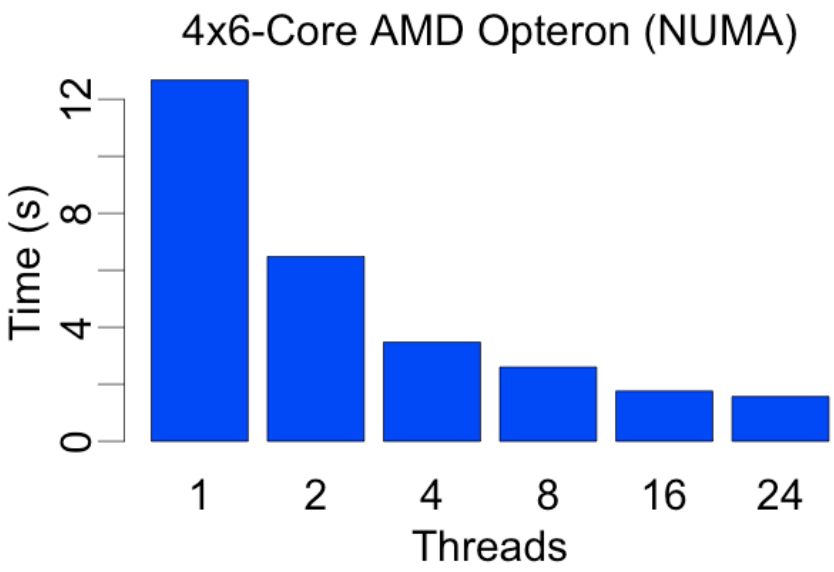


smottish@sfu.ca

Results: “Lock-step”

- Run-times

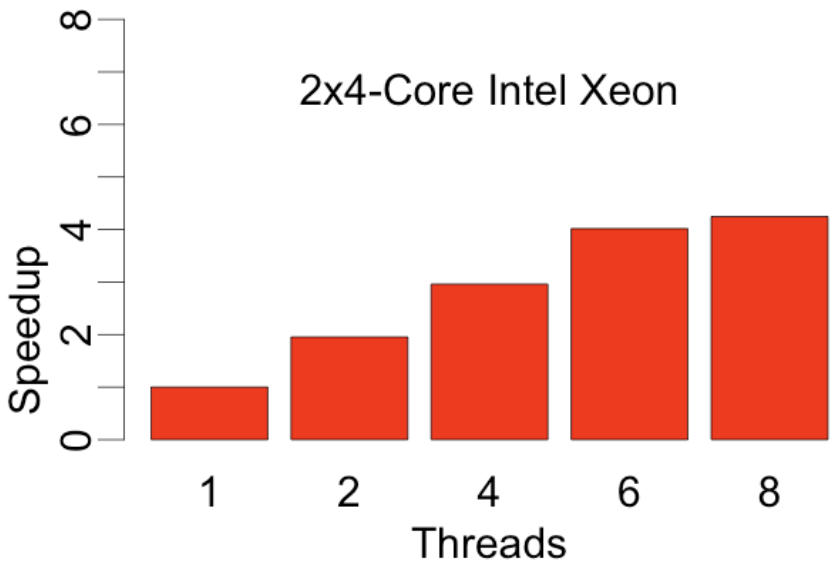


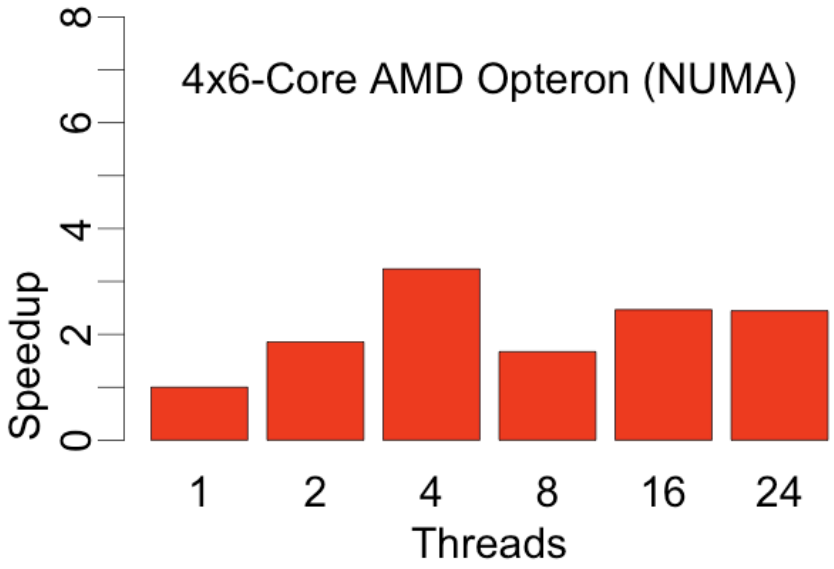


smottish@sfu.ca

Results: “Message Passing”

- Speed-up

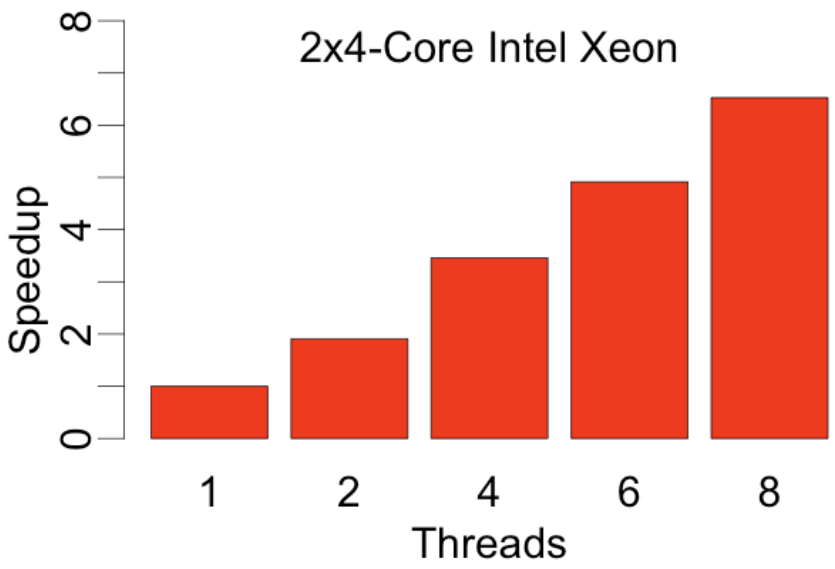


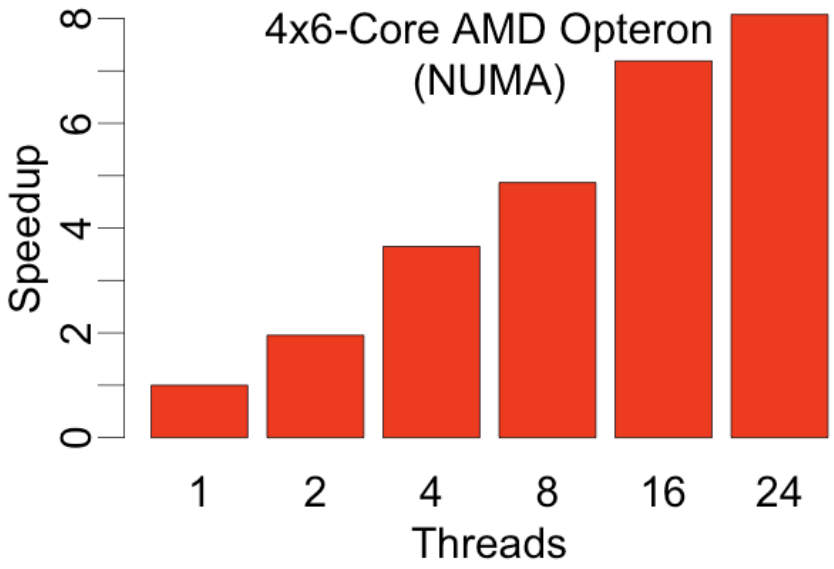


smottish@sfu.ca

Results: “Lock-step”

- Speed-up





smottish@sfu.ca

Conclusion

- Achieved a maximum speedup of 8.08
- Managing data and inter-processor communication is crucial!
- Further optimization can be done to further reduce:
  - Memory latencies
  - Inter-processor communication
- Implement and compare a GPU version
- Hybrid CPU-GPU version

smottish@sfu.ca