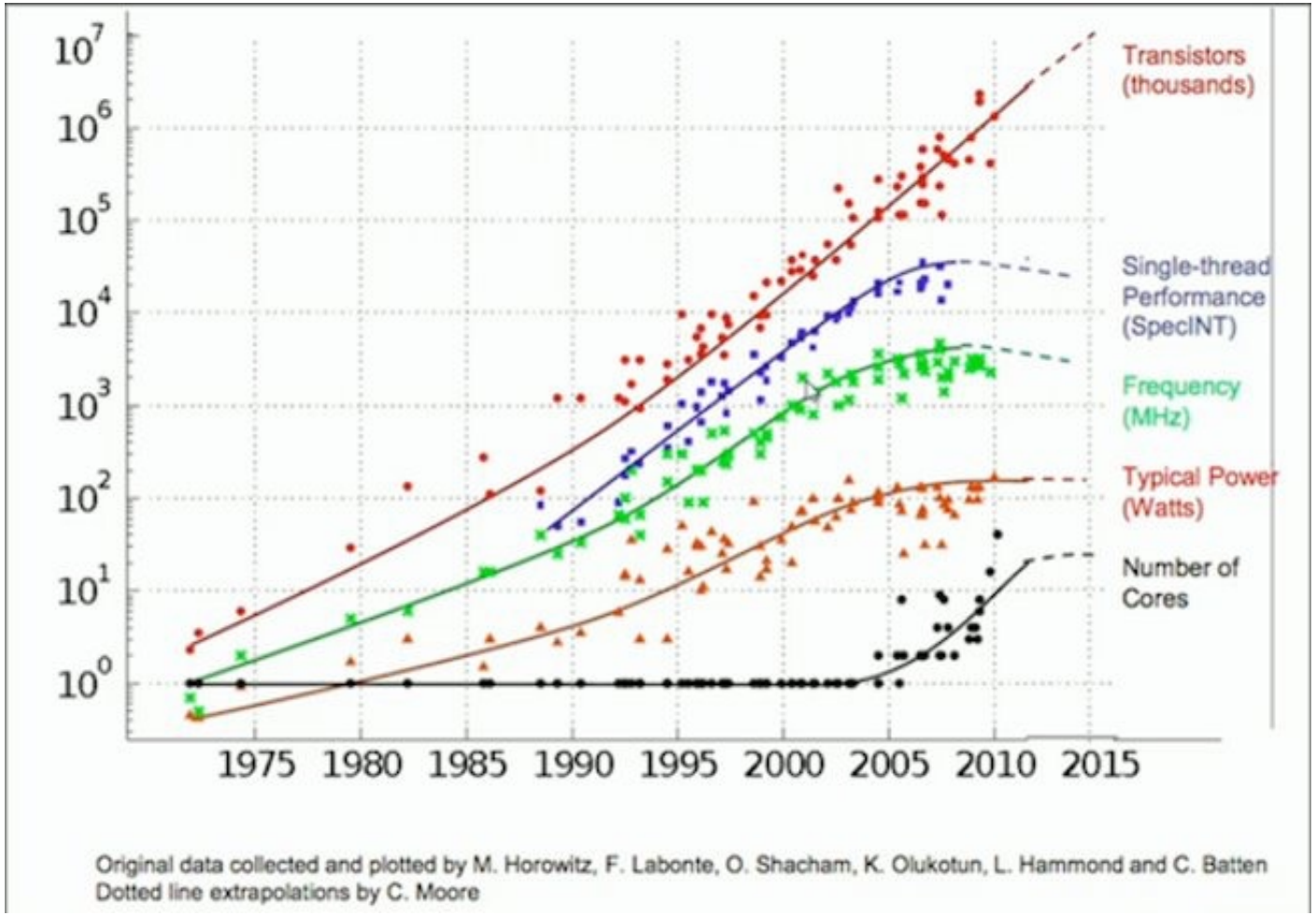
Manycore systems

Hardware Design Trends

- Single core performance is almost saturated
- Trend towards manycore
  - Many cores on same chip

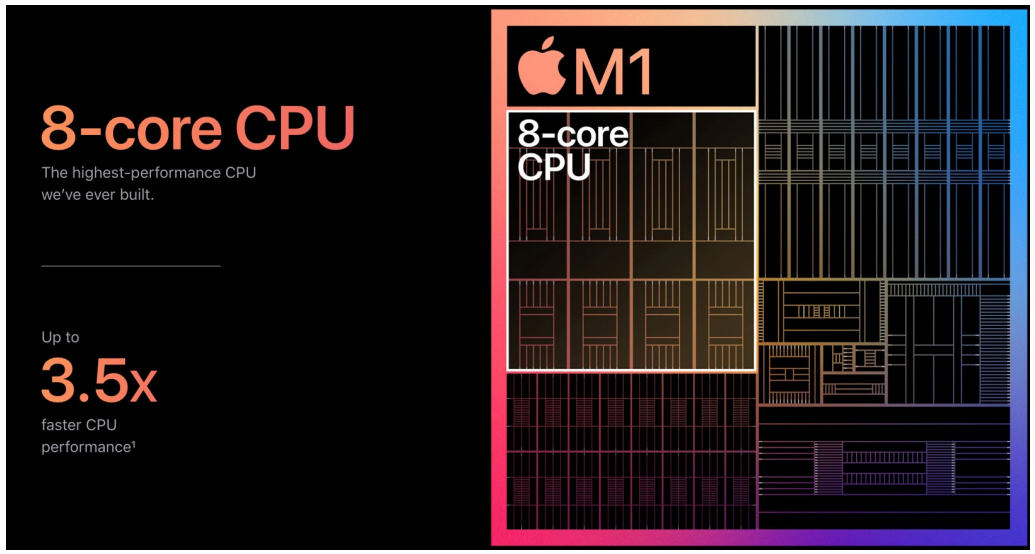


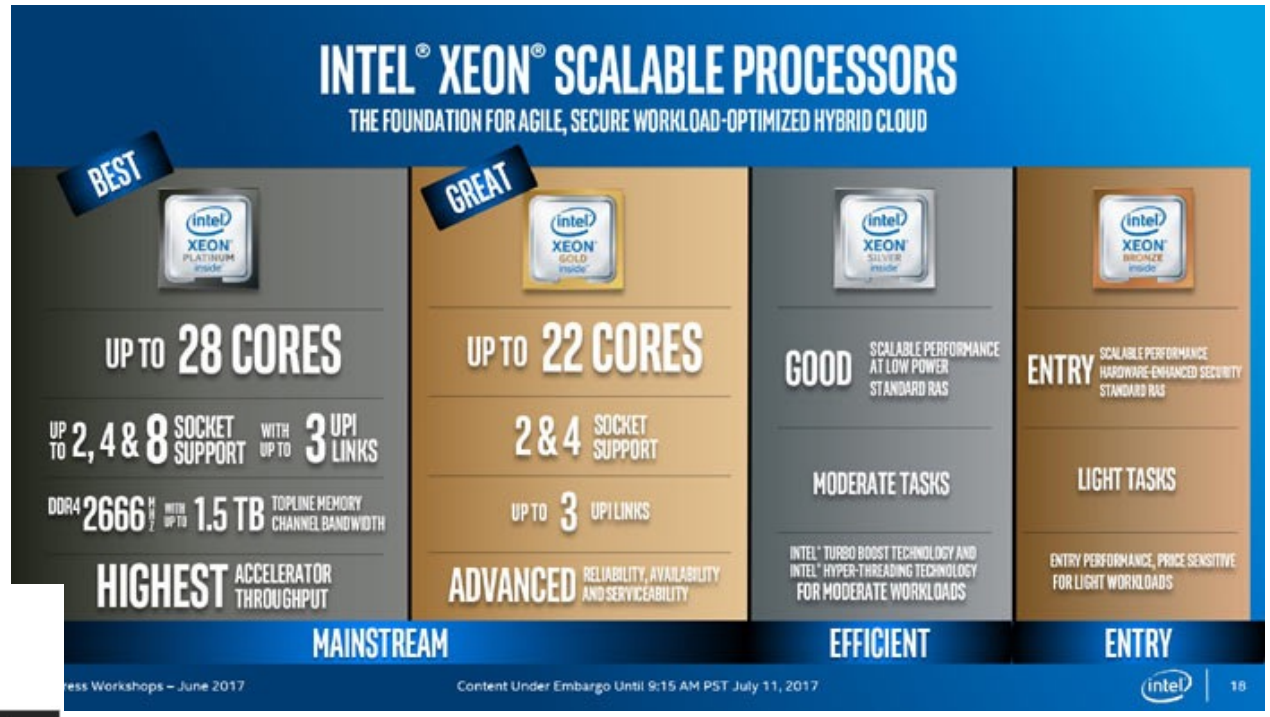
http://meseec.ce.rit.edu/551-projects/fall2013/3-2.pdf

Single-core Processors

- Power wall
  - Billions of transistors
  - Low power budget for mobile, wearables
- Frequency wall
  - $P=cv^2f$, Contributing to power
  - Design complexity
  - More area
- Too much expense for too little return

Real-world manycore examples





TESLA V100

21B transistors
815 mm^2

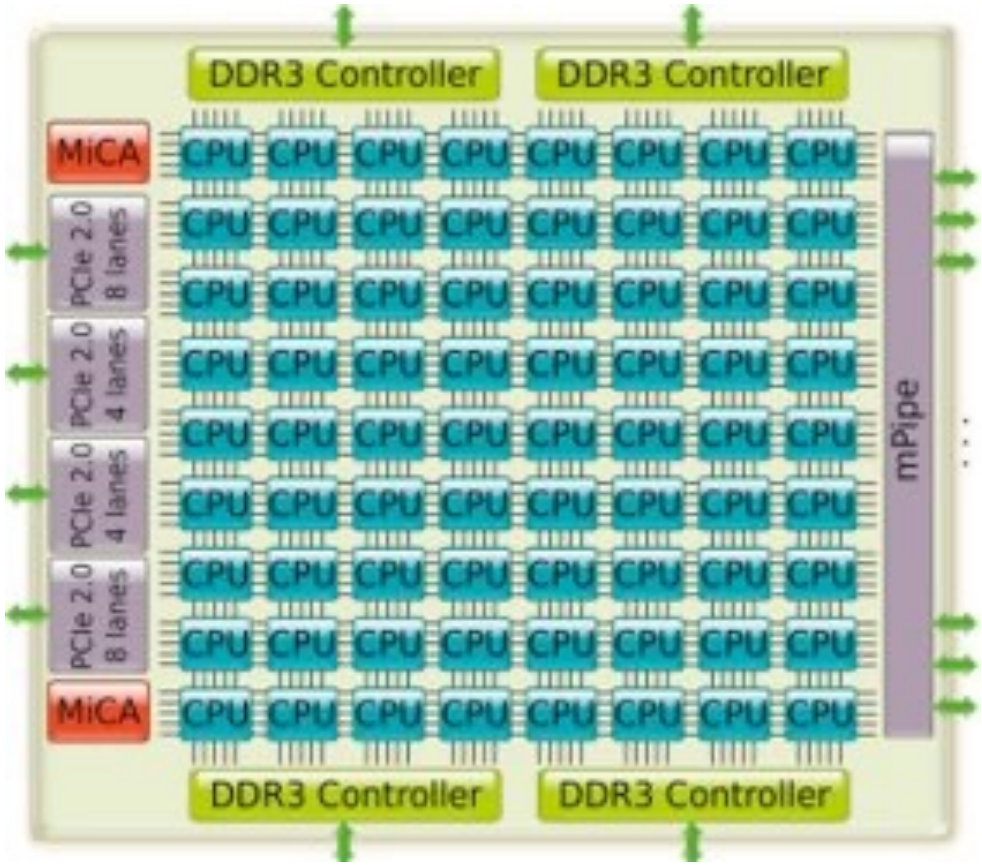
80 SM
5120 CUDA Cores
640 Tensor Cores

16 GB HBM2
900 GB/s HBM2
300 GB/s NVLink

*full GV100 chip contains 84 SMs

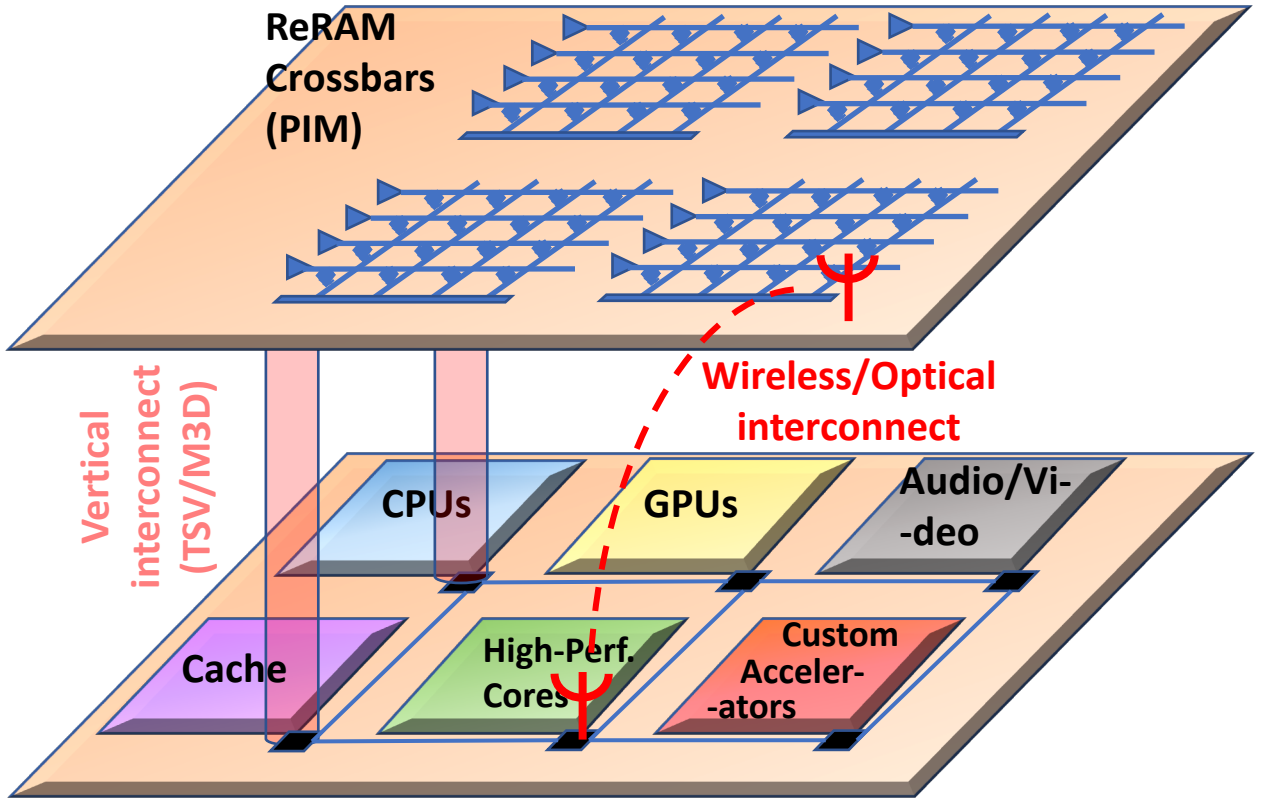
Manycore Processors

- **Few examples:**
  - **ZettaScaler**
  - **Tilera**
  - **Kalray**
  - **TrueNorth**
  - **Intel Xeon**
  - **Intel M1**

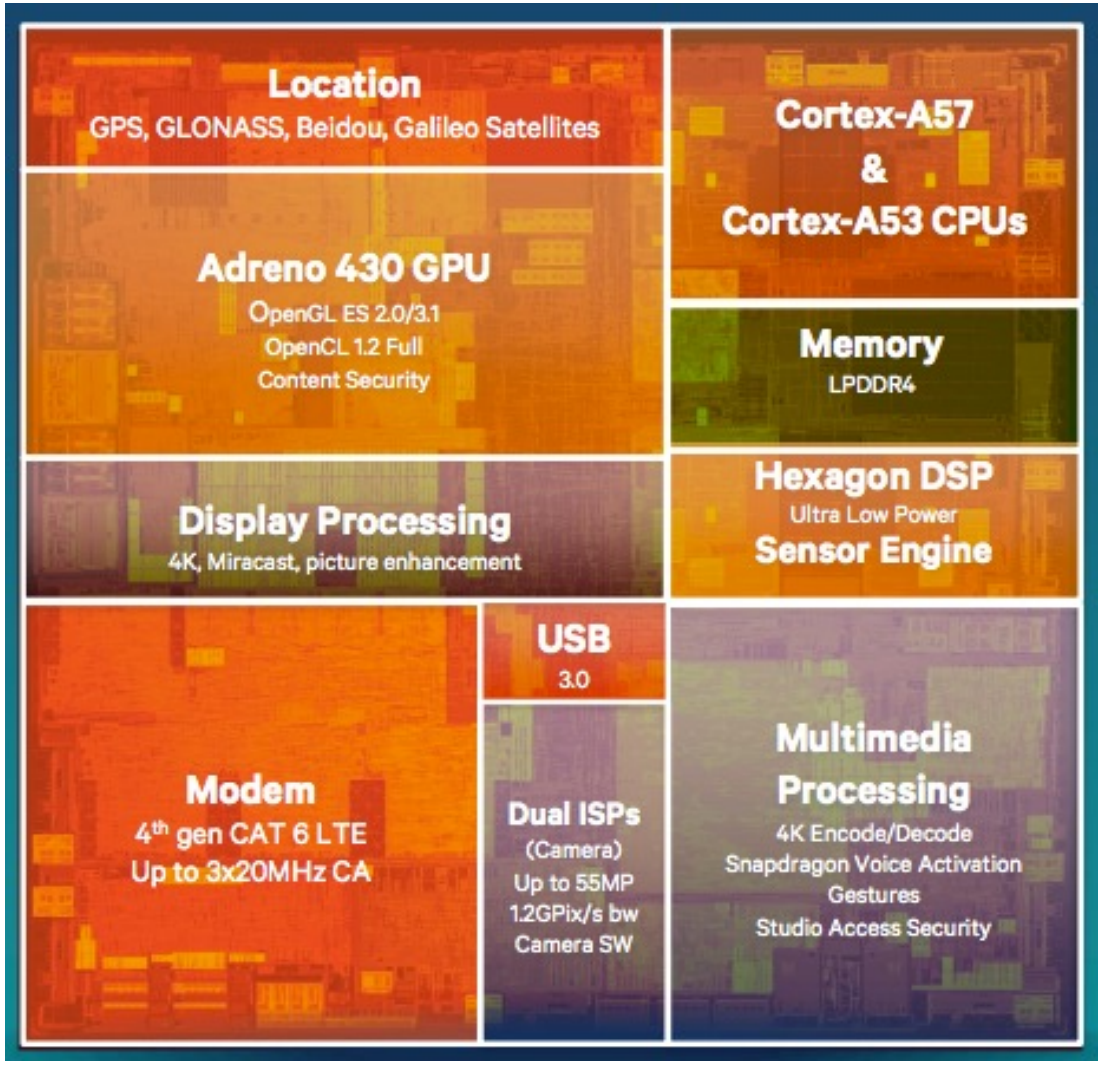


https://en.wikipedia.org/wiki/Manycore_processor

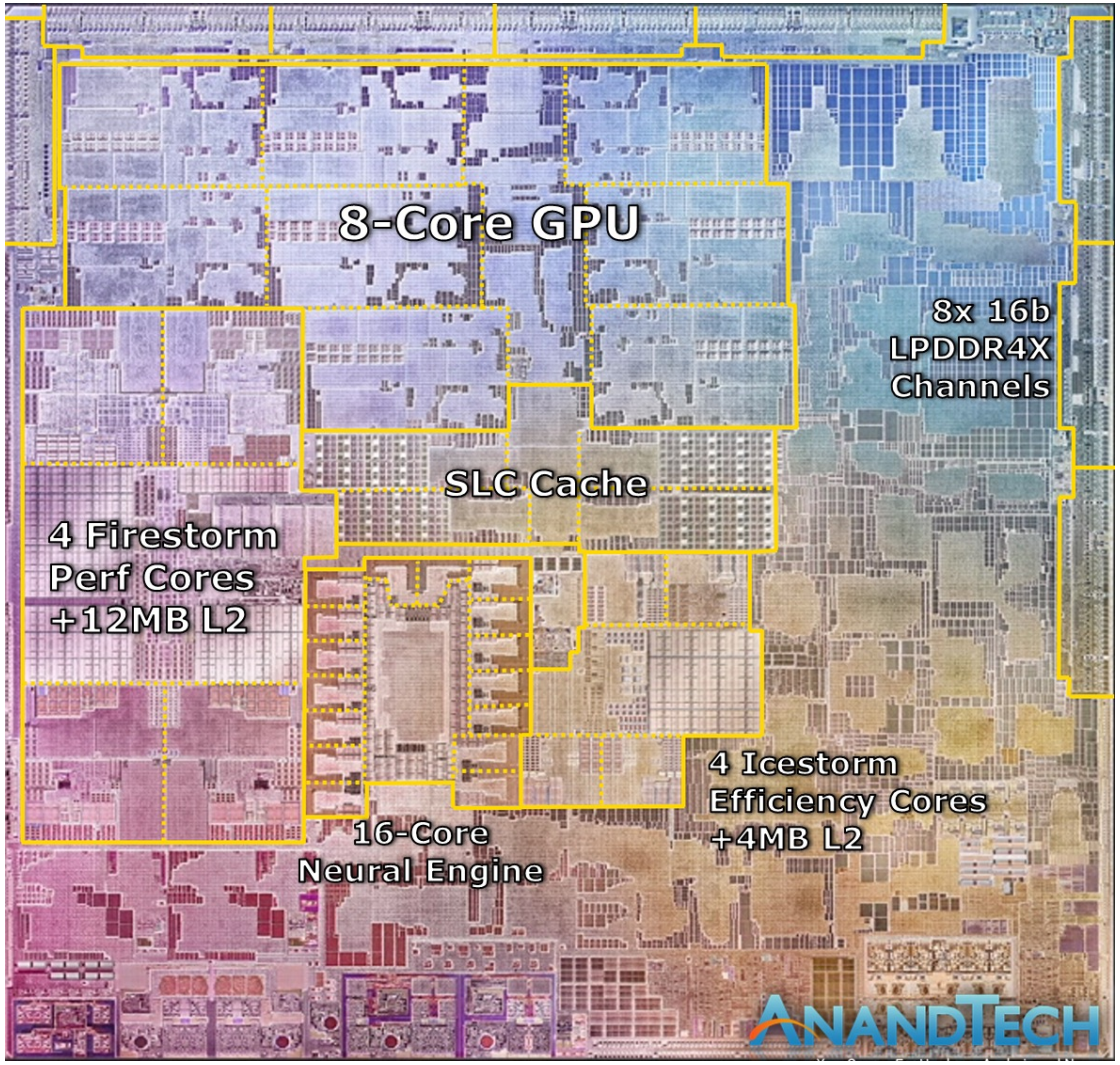
Heterogeneous Manycore Processors



Real-world heterogeneous manycore examples

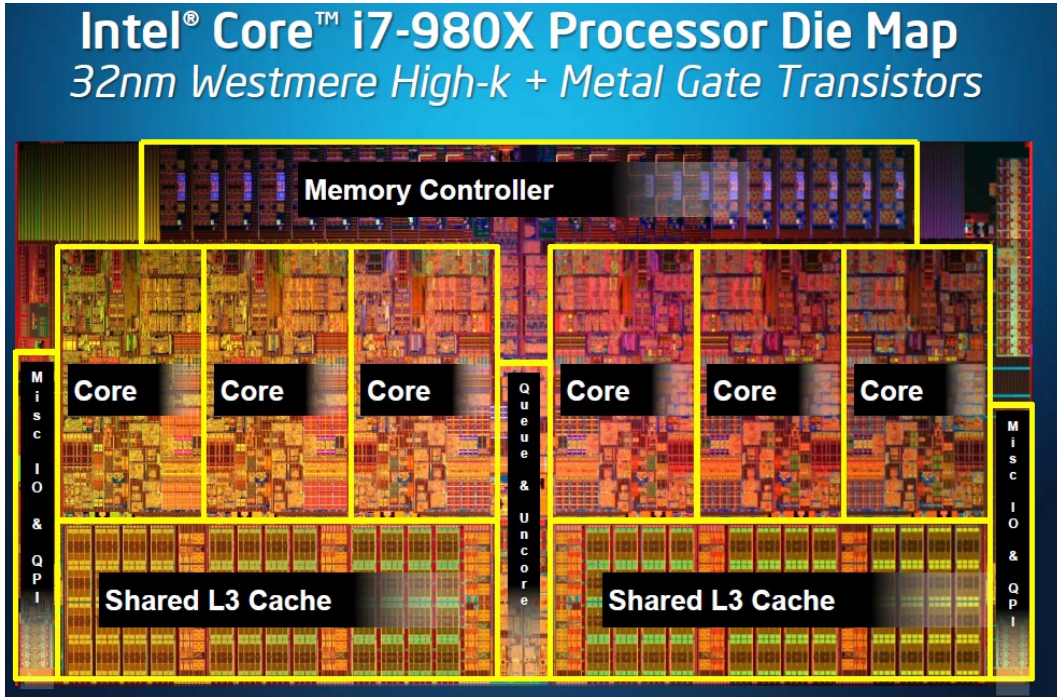


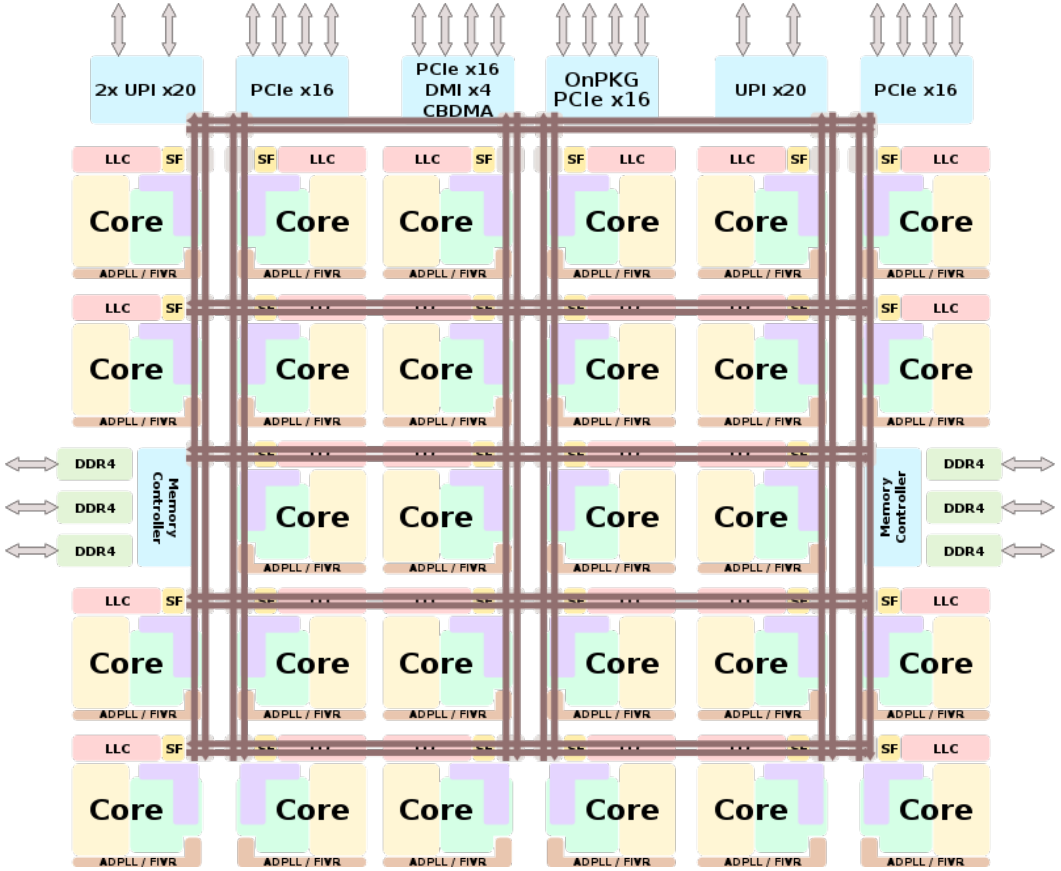
Qualcomm Snapdragon



Apple M1

Multi-core vs Manycore Processors

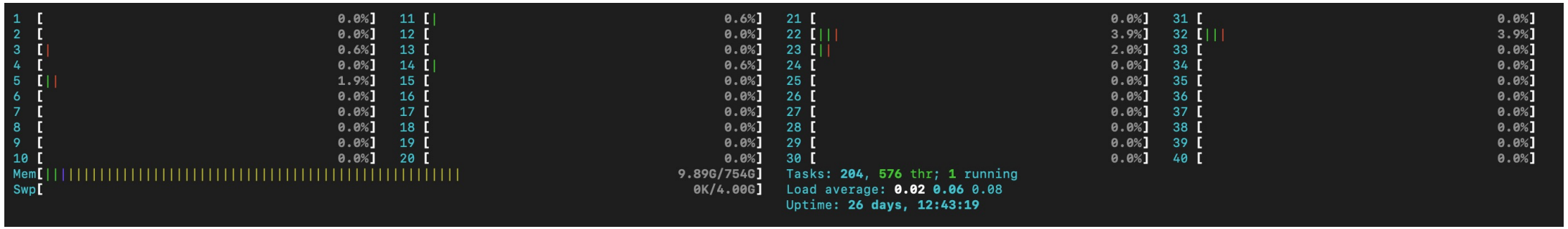




Intel Xeon

- Usually more that 10 cores are called Manycore
- Intel i7 (6-cores) vs Intel Xeon (28 cores)

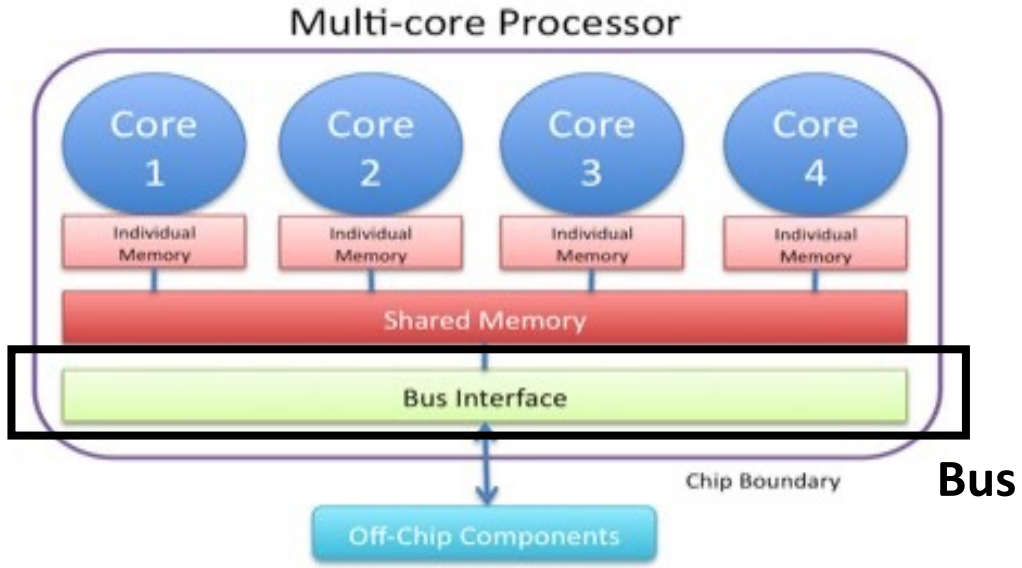
Manycore Processor example



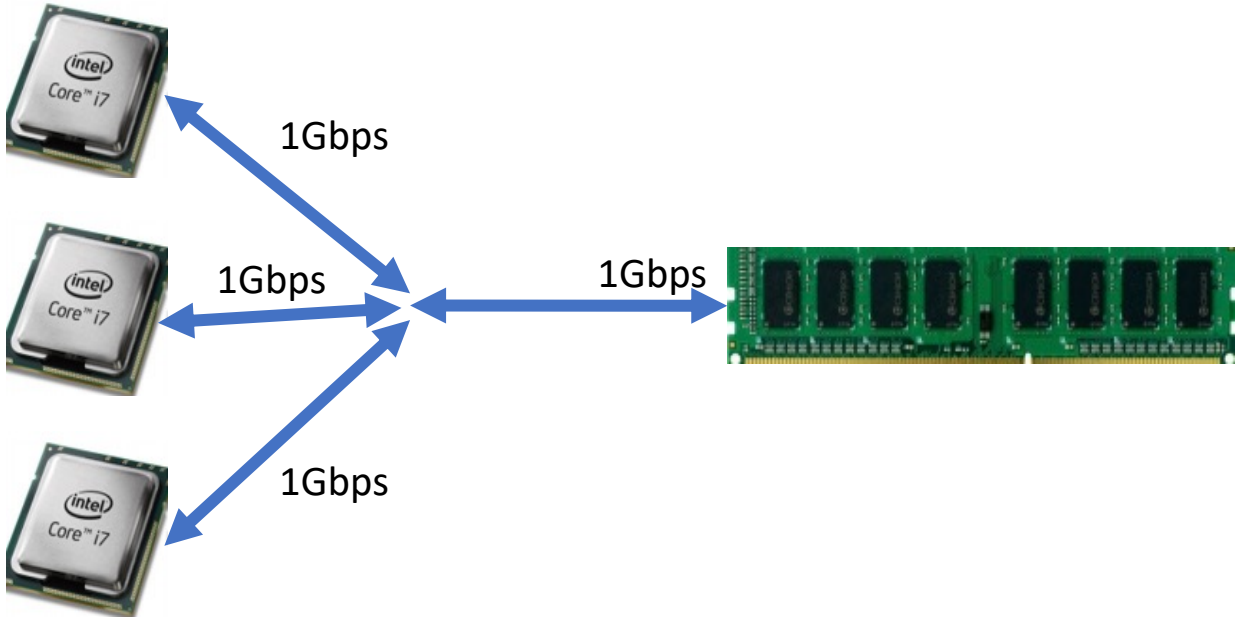
Intel(R) Xeon(R) Gold 5115 CPU @ 2.40GHz

- **40 CPUs in Intel Xeon (Available at Duke)**
- **System information available by using following commands:**
  - **htop, lscpu, etc.**

Manycore challenges: Hardware



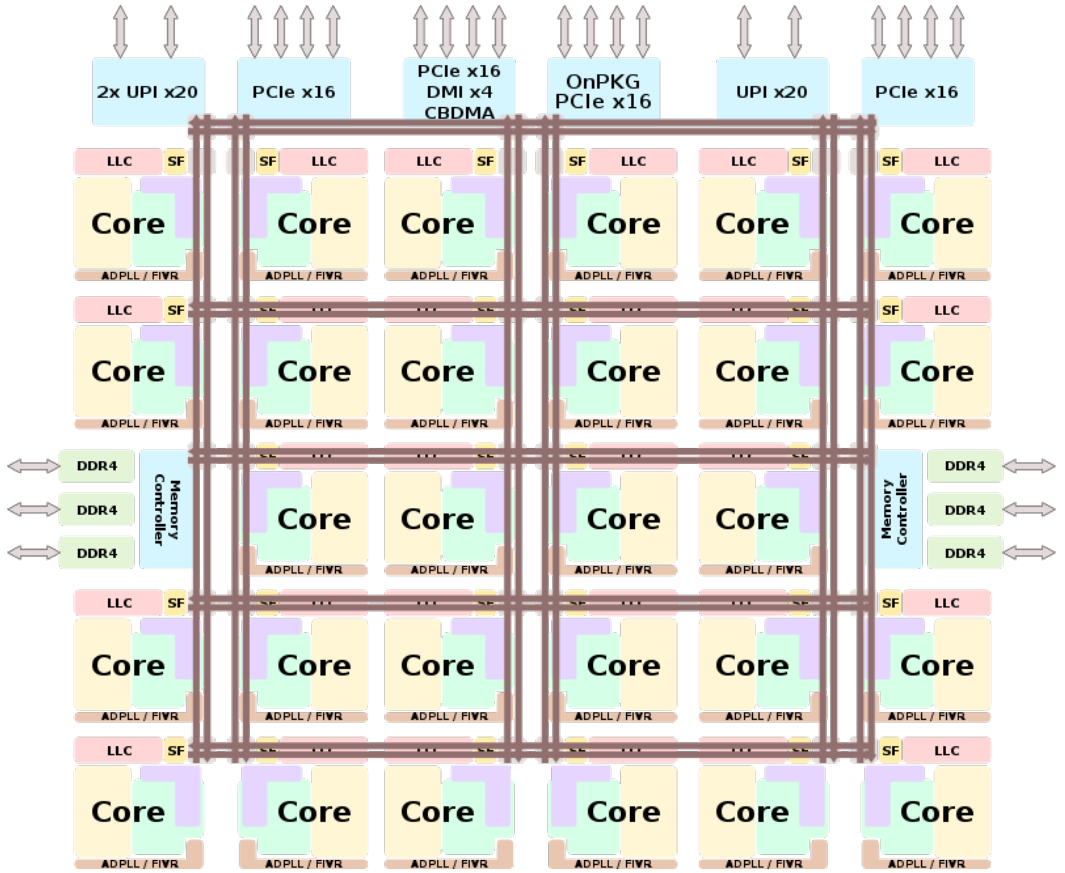
Bus



- **Communication bottleneck**
  - **How does so many cores communicate?**
- **Memory bottleneck**
  - **High throughput requirements**

Manycore challenges: Software

- Run multiple tasks at the same time
- Software optimization
  - How do you use so many cores?
  - No point using serial code



Multi-thread programming

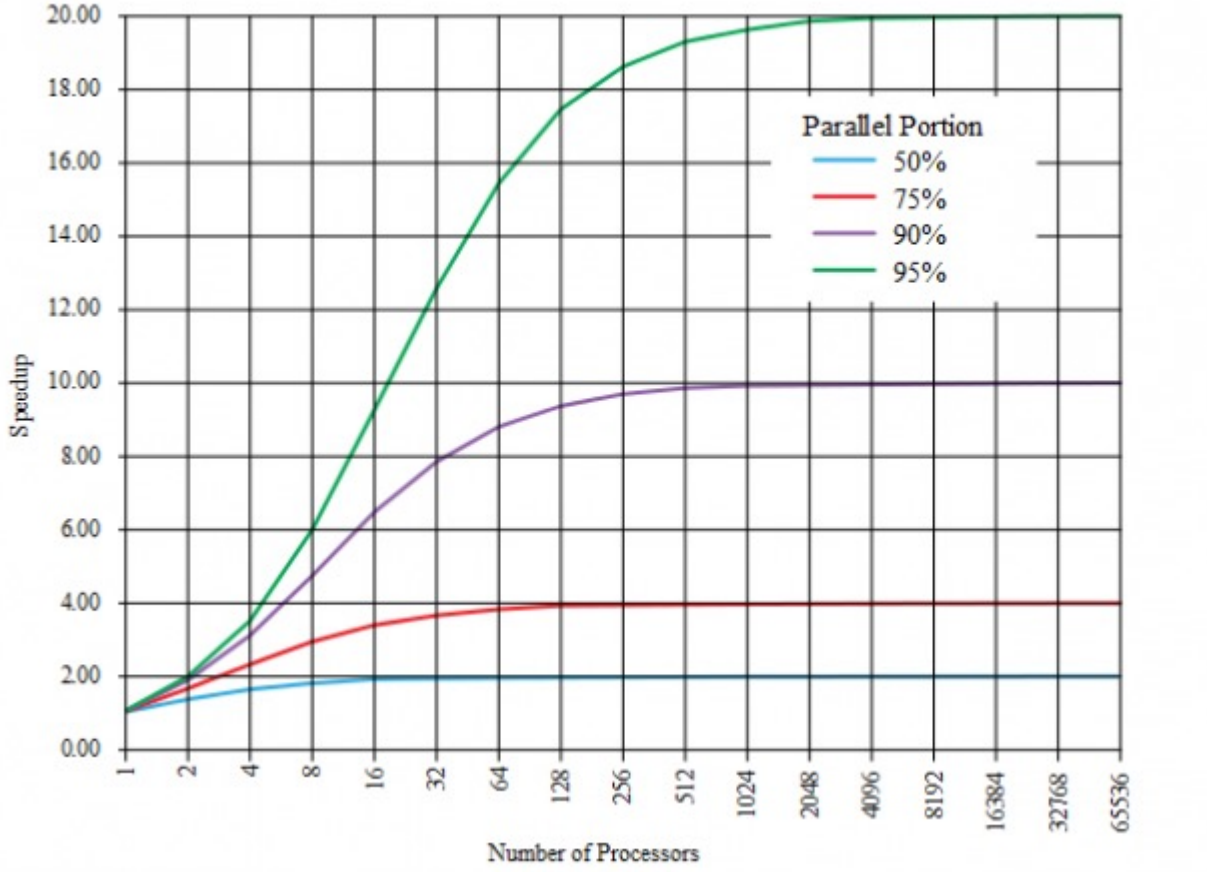
```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
using namespace std;
#define NUM_THREADS 5
void *PrintHello(void *threadid) {
   long tid;
   tid = (long)threadid;
   printf("Hello World! Thread ID, %d\n", tid);
   pthread_exit(NULL);
}
int main () {
   pthread_t threads[NUM_THREADS];
   int rc;
   int i;
   for( i = 0; i < NUM_THREADS; i++ ) {
      cout << "main() : creating thread, " << i << endl;
      rc = pthread_create(&threads[i], NULL, PrintHello, (void *)i);
      if (rc) {
         printf("Error:unable to create thread, %d\n", rc);
         exit(-1);
      }
   }
   pthread_exit(NULL);
}
```

```
$gcc test.cpp -lpthread
$./a.out
main() : creating thread, 0
main() : creating thread, 1
main() : creating thread, 2
main() : creating thread, 3
main() : creating thread, 4
Hello World! Thread ID, 0
Hello World! Thread ID, 1
Hello World! Thread ID, 2
Hello World! Thread ID, 3
Hello World! Thread ID, 4
```

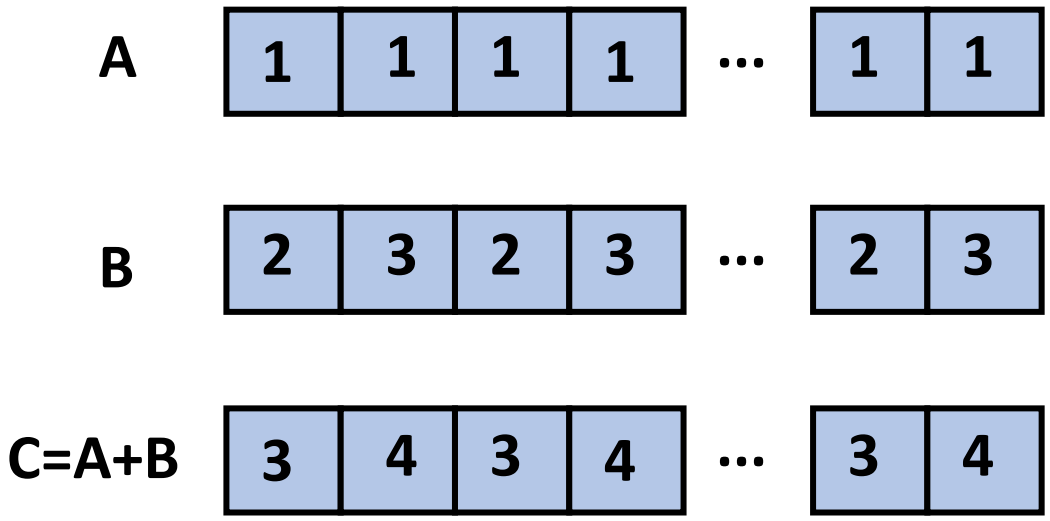
https://www.tutorialspoint.com/multithreading-in-c

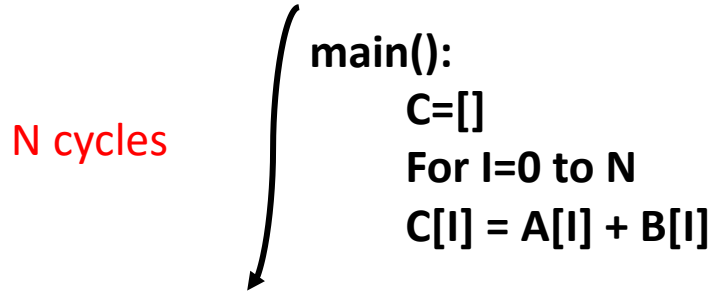
Challenges with Multi-threading

- Not all parts of code can be parallelized
  - Atomic operations
  - I/O operations
- Identifying parallelizability left to the programmer
- New algorithms are needed

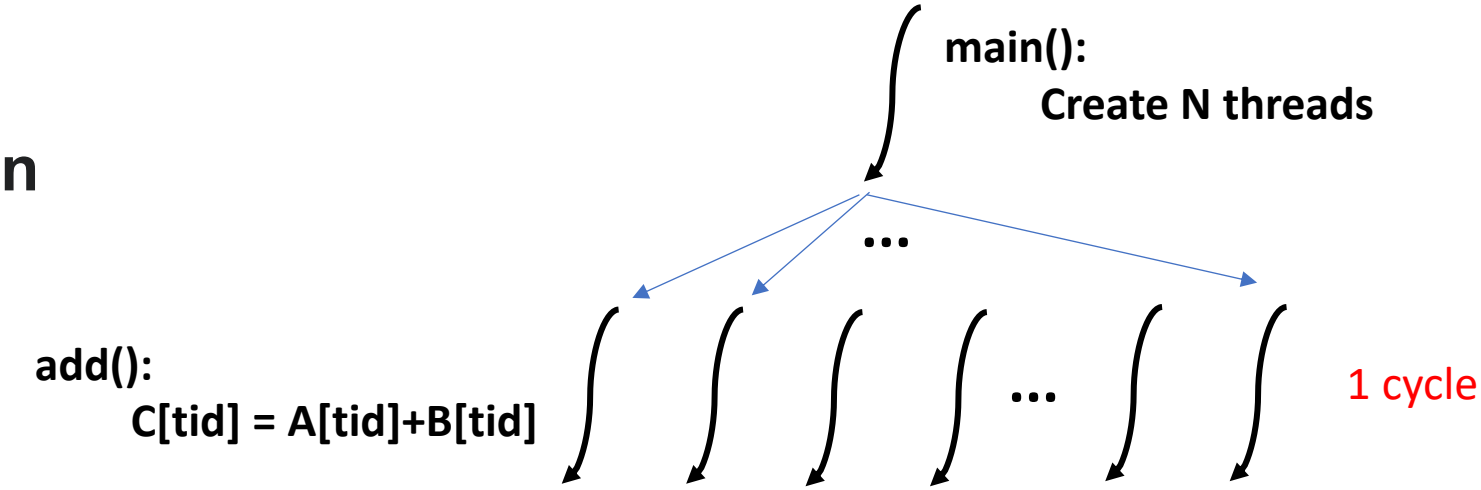


Example: Parallel add





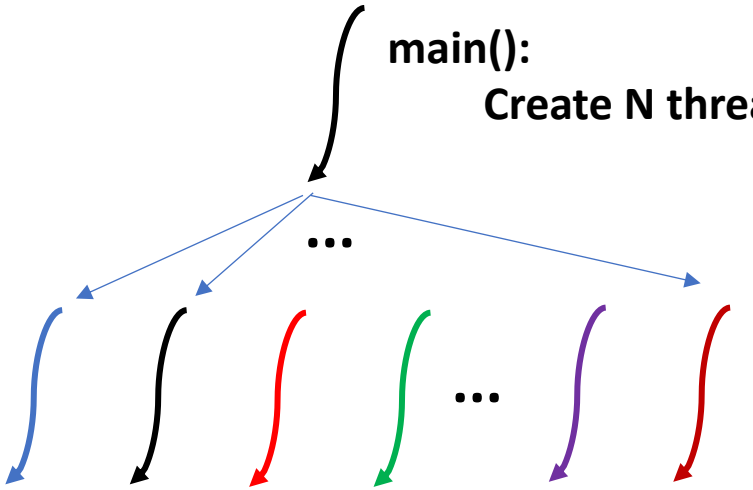
- Significantly faster addition
  - Parallel execution

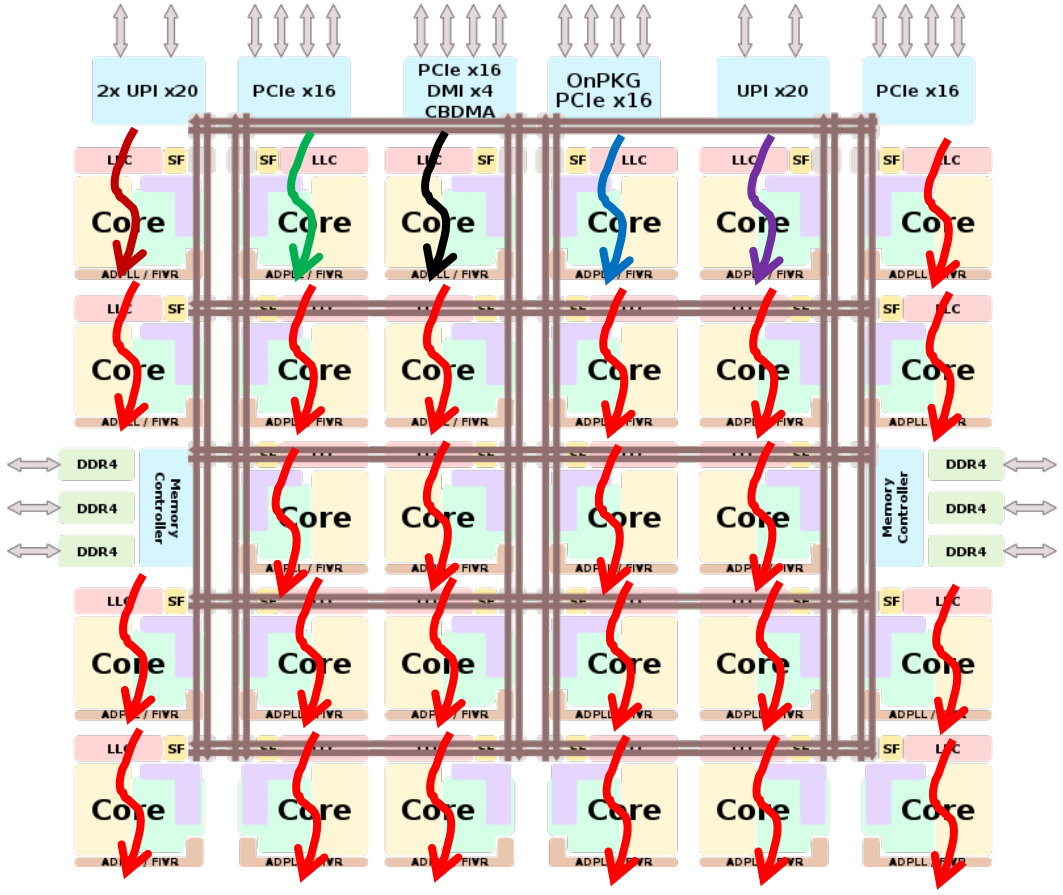


Task mapping to manycore

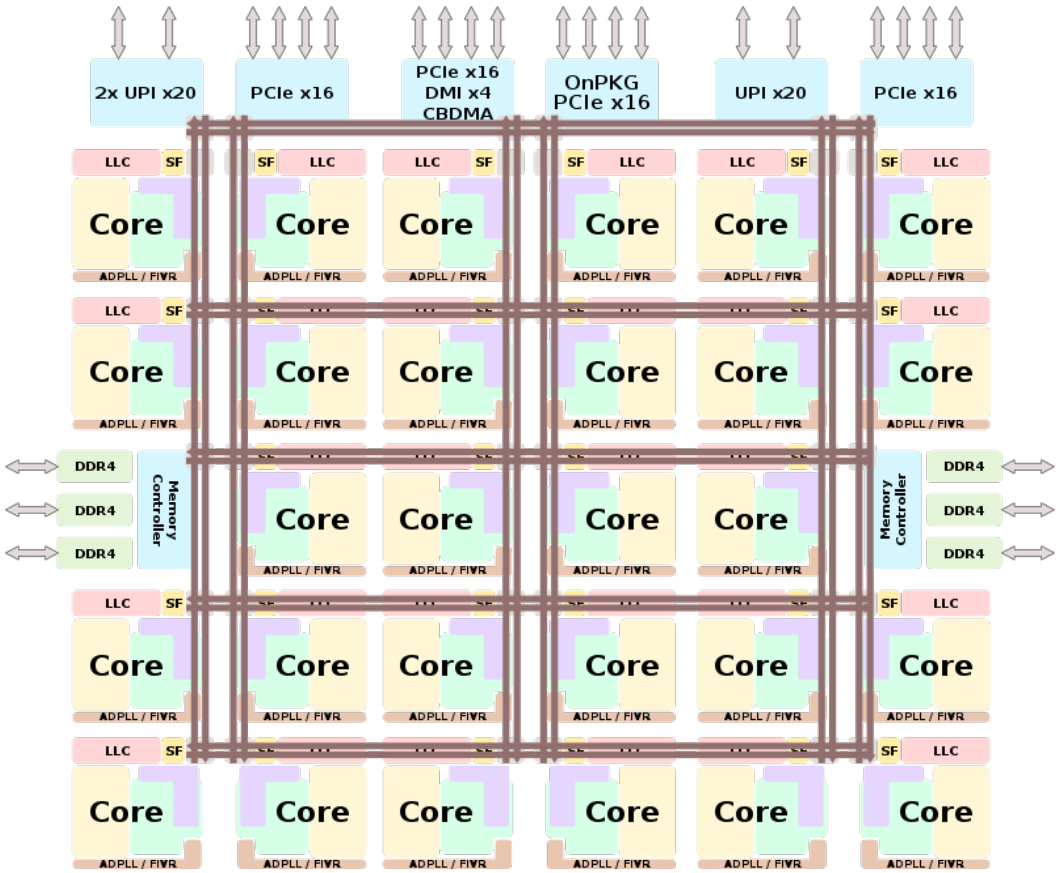
main():
Create N threads

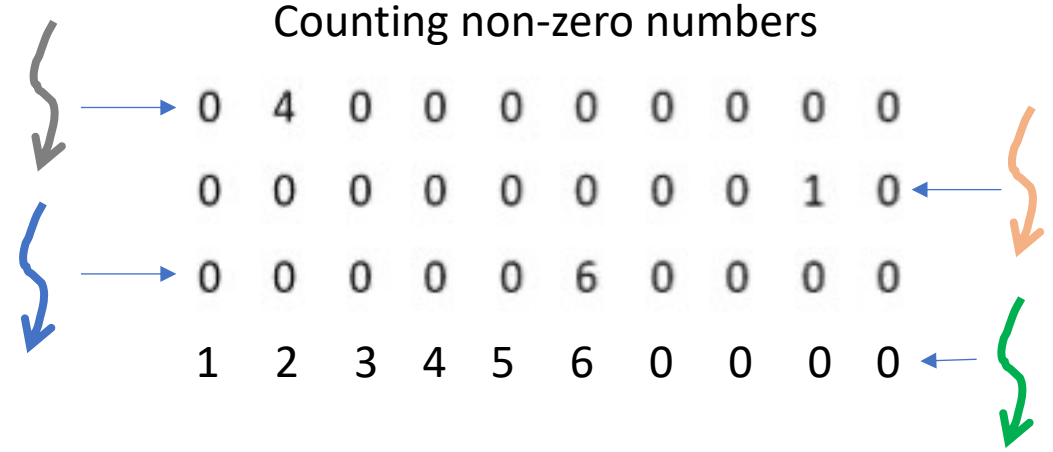
add():
C[tid] = A[tid]+B[tid]





Why task mapping?

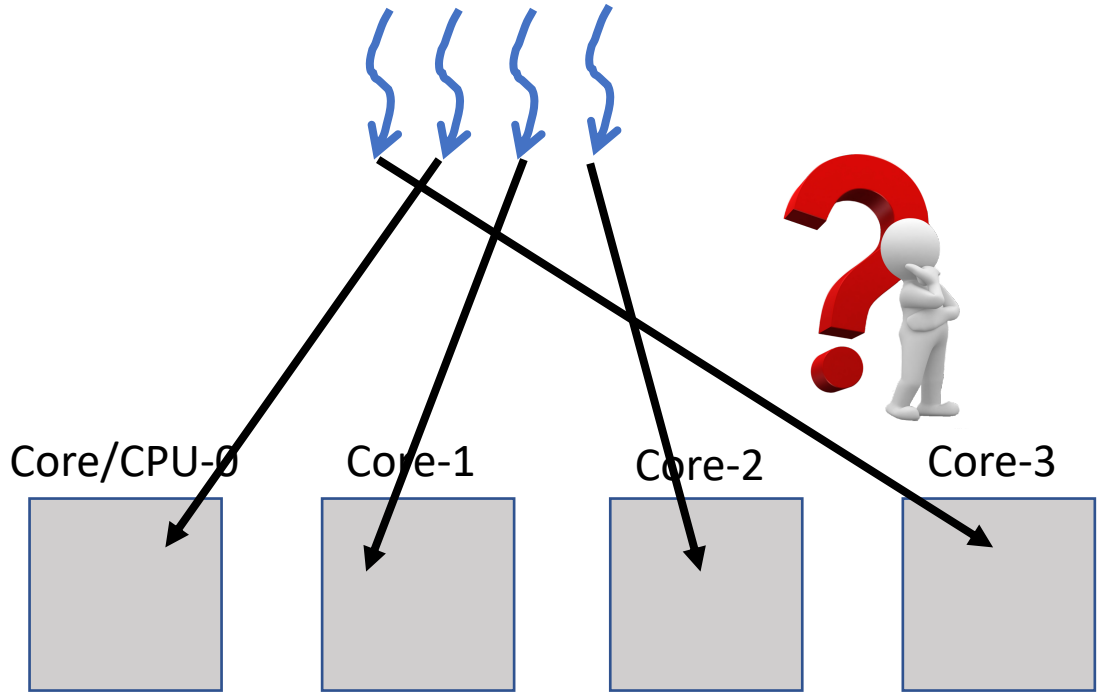




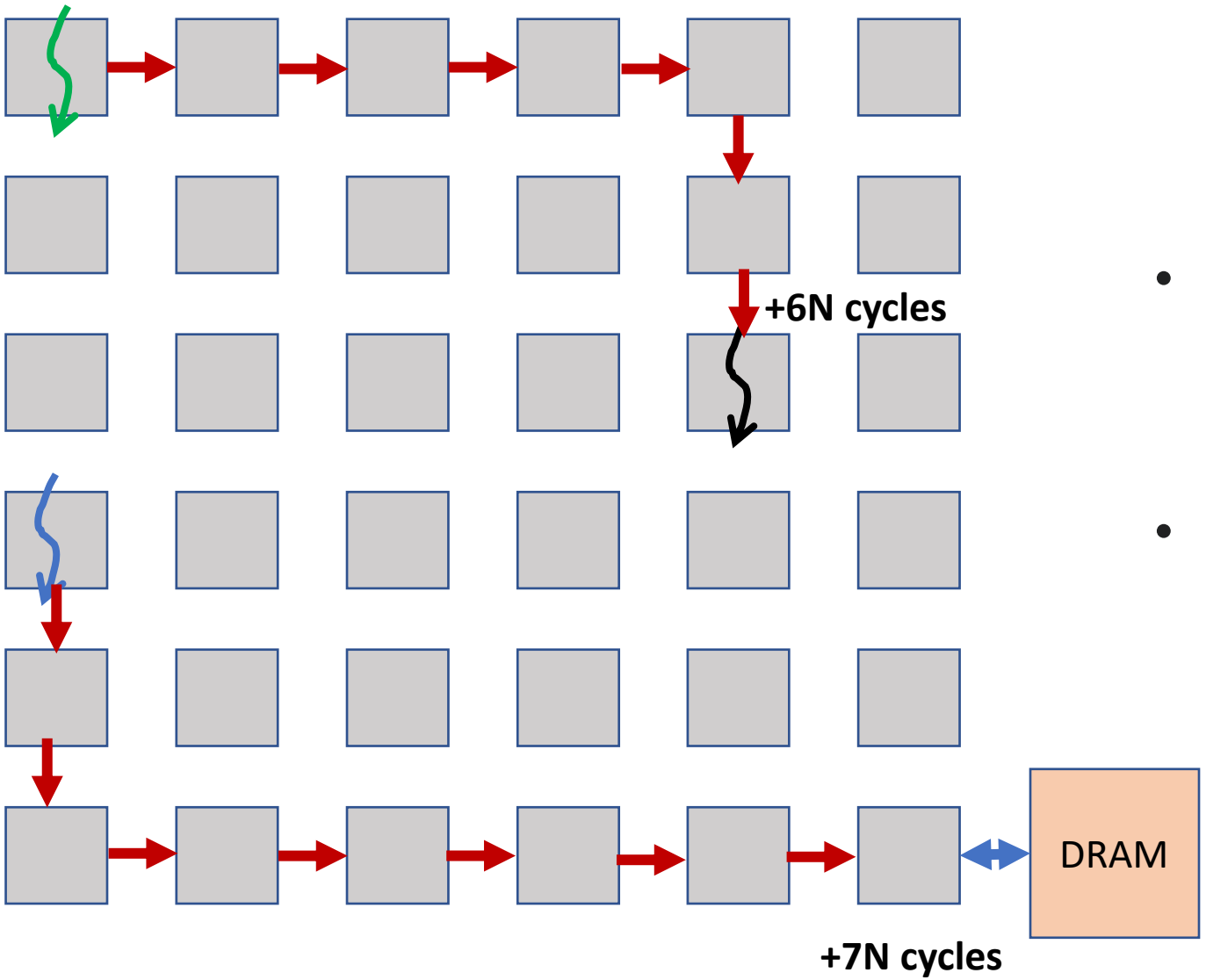
- Threads perform unequal tasks
  - Execution time
  - Communication
  - Thermal
  - How to map?

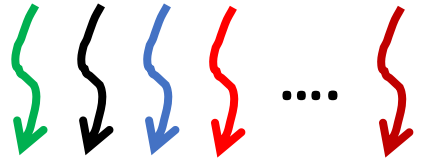
Task Mapping

- How to assign threads to cores?
  - Execution time
  - Communication
  - Power
  - Temperature



Mapping in manycore

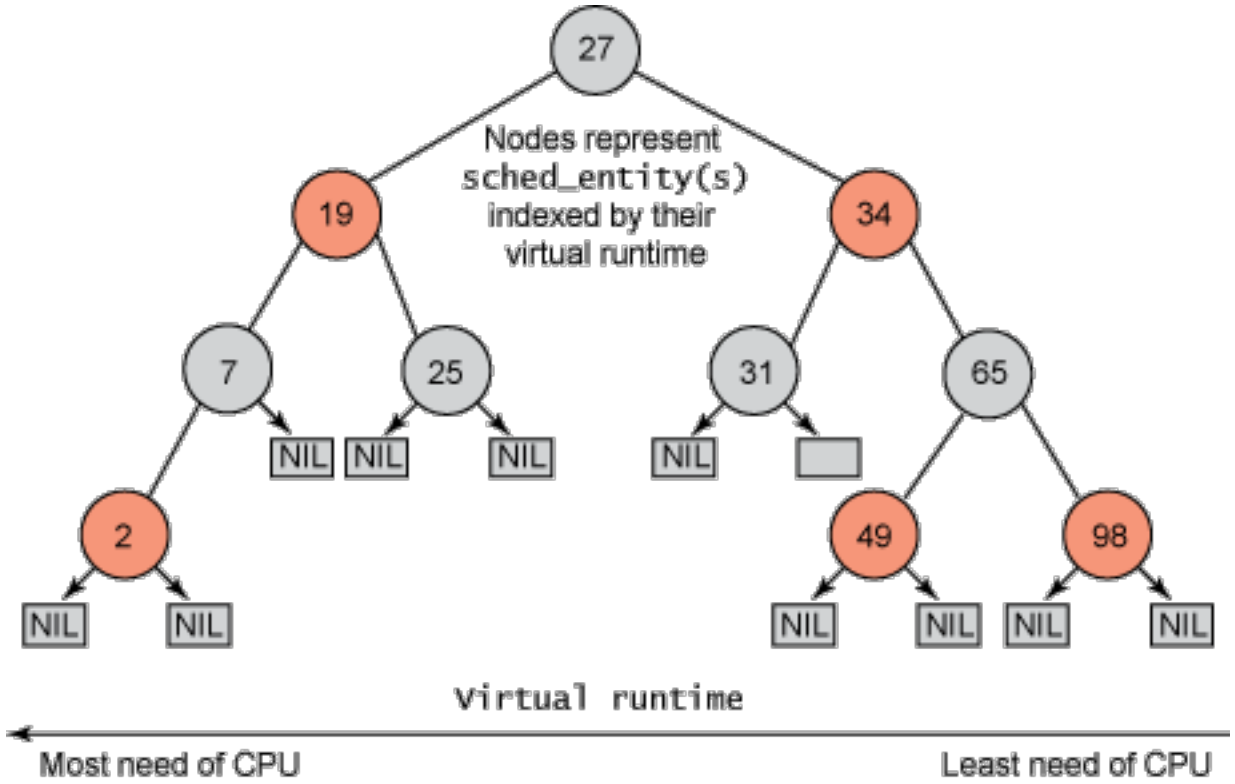




- N tasks → N cores
  - Communication
    - Performance/Execution time
  - Thermal
- N! (N factorial) cases
  - Impossible to explore all possible solutions
  - How to map these tasks?

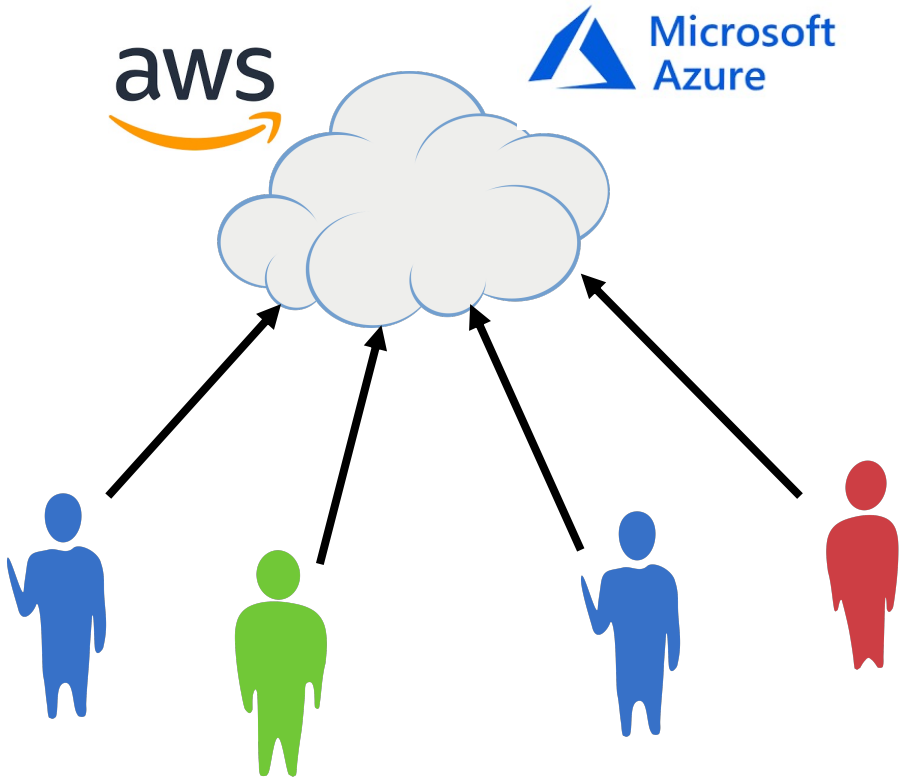
Completely Fair Scheduler: Linux

- Available with Linux
- Aims to maximize CPU utilization
- https://developer.ibm.com/tutorials/l-completely-fair-scheduler/
- We're doing application-specific task mapping



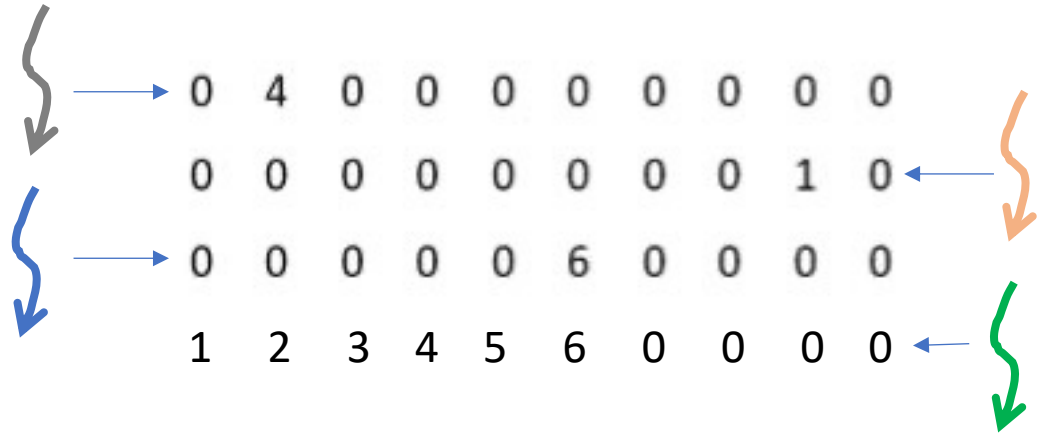
Task mapping in datacenters

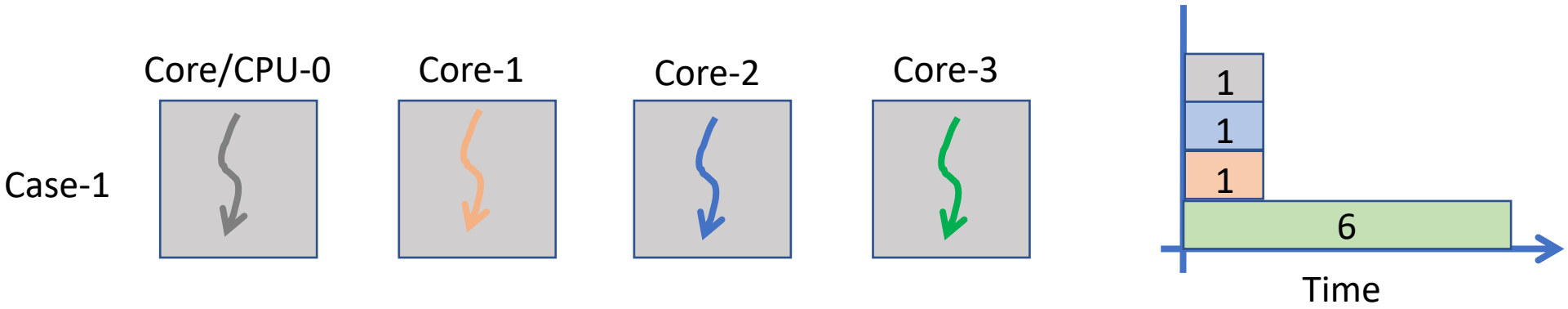
- Cloud service must provide High performance
- Customers don’t wait to pay more
  - Execute in least time possible
- Providers want to serve many customers
  - Allocate and utilize resources efficiently



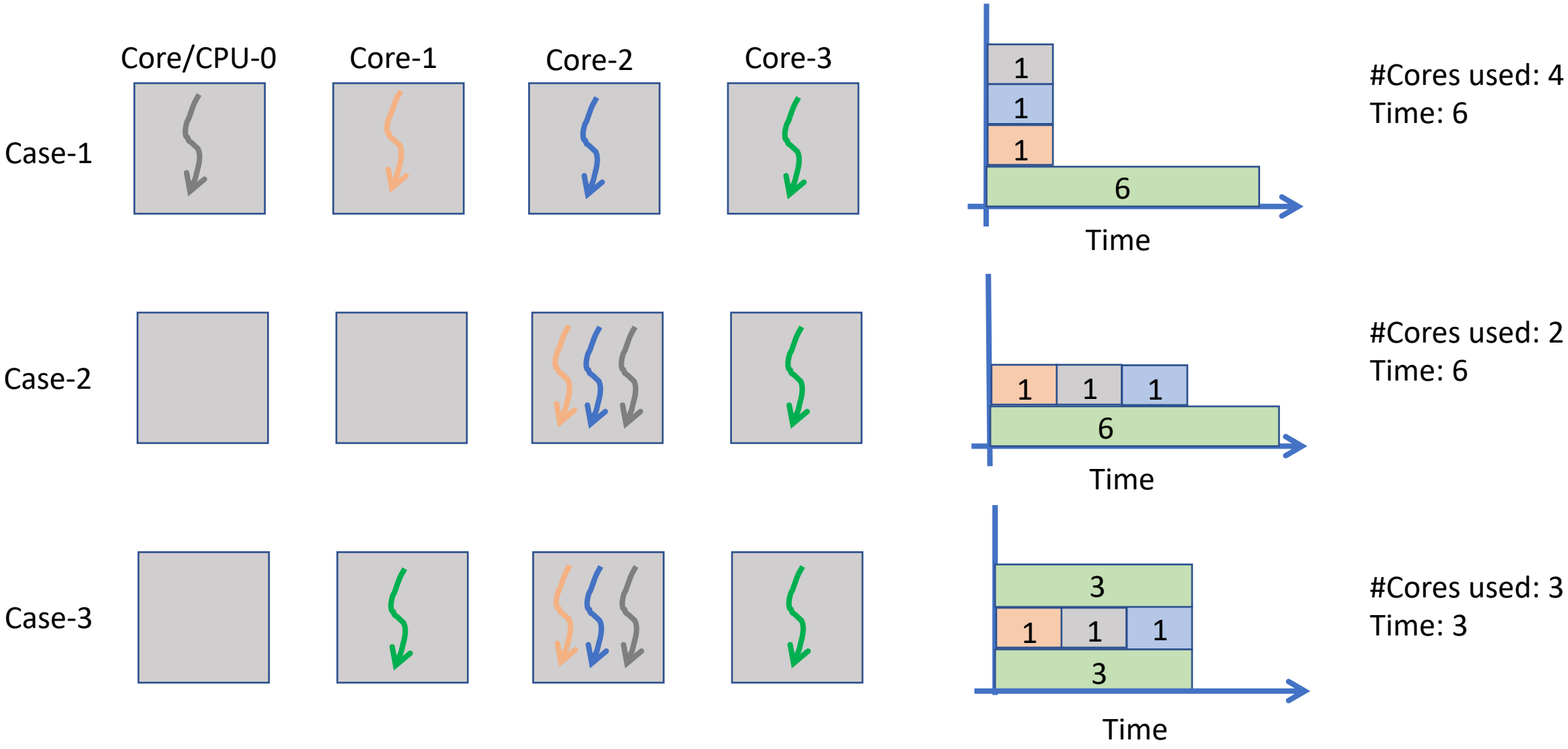
Load balancing

- Threads have unequal work
- Task scheduling
- Examples:
  - Sparse matrix operations
  - Graph applications
  - Counting

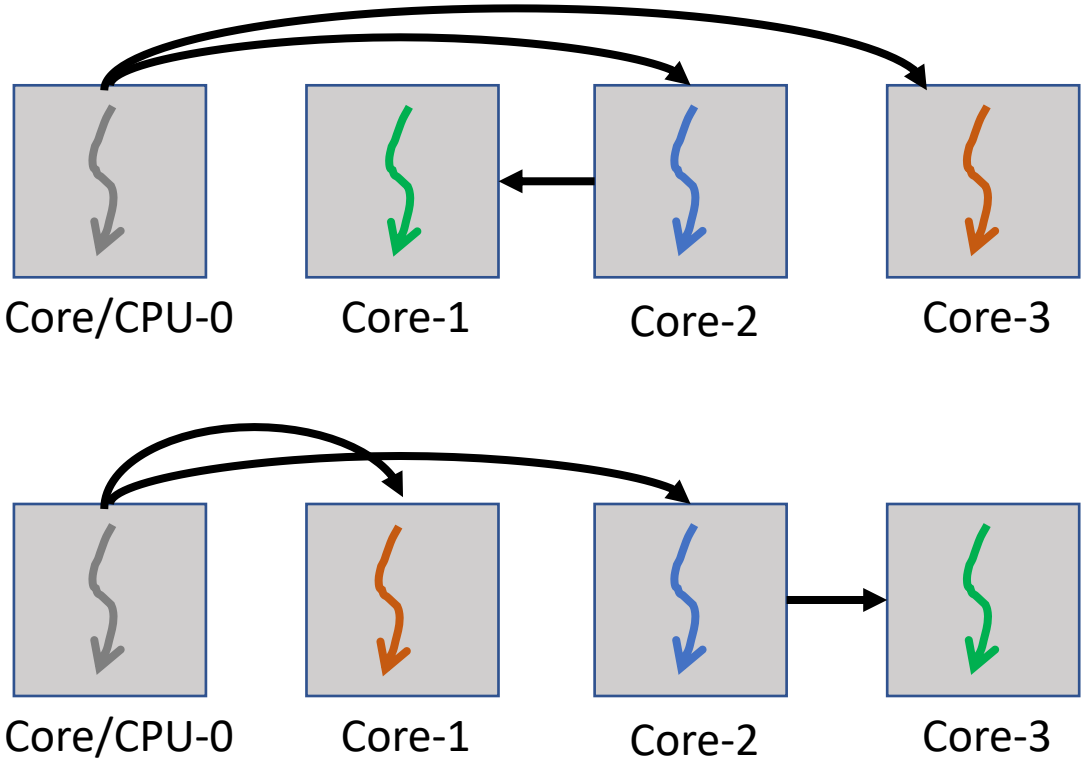


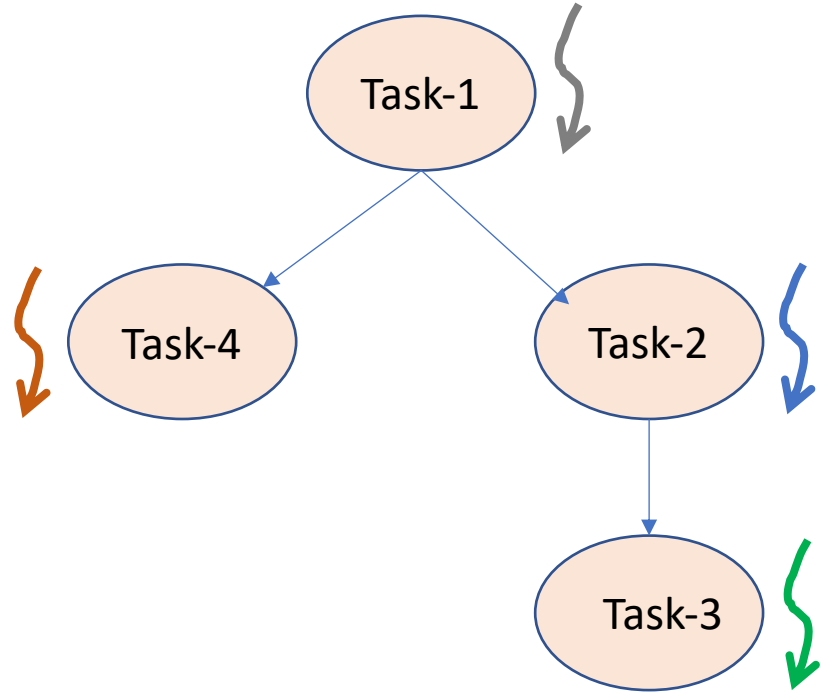


Effect of mapping



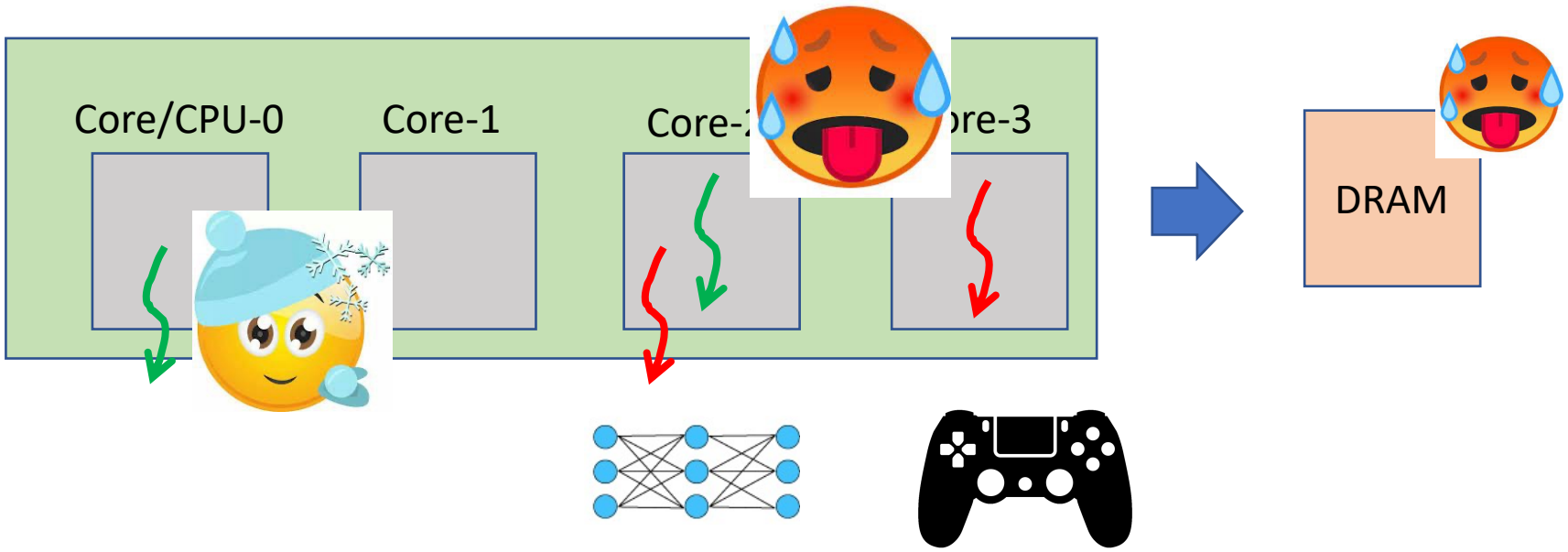
Communication





- **Map using a Task graph**
- **Common in many workloads**
  - **Input dependence**
  - **Leads to communication**

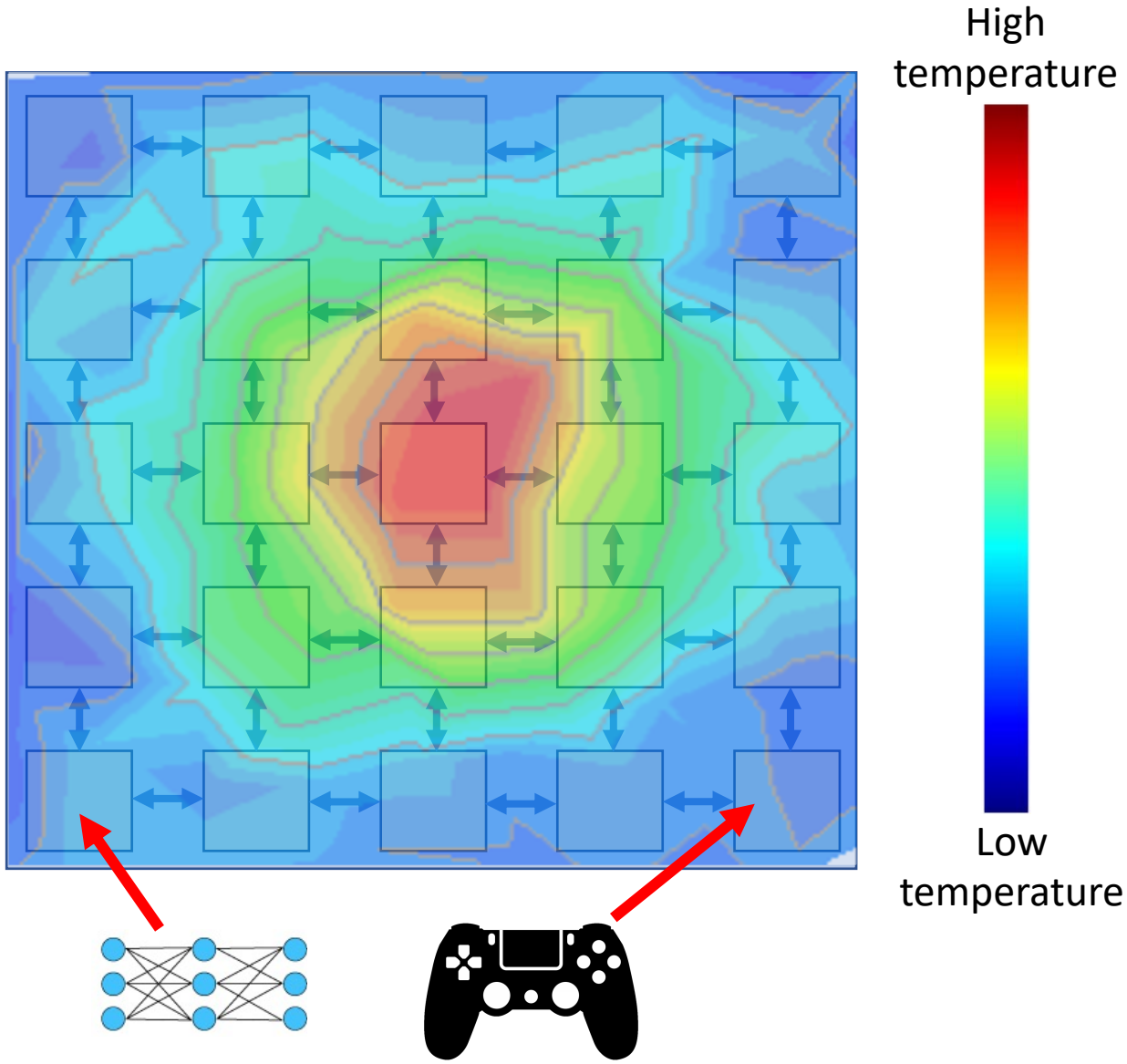
Thermal concerns



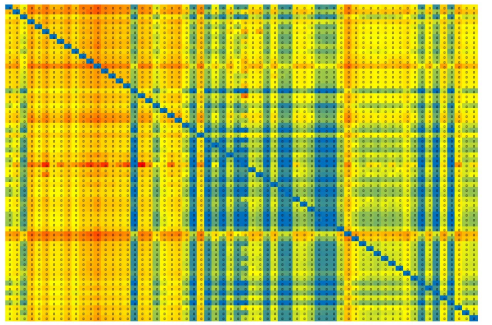
- Computationally intensive workloads
  - Mapping them close together will lead to heat in the vicinity
- Many hardware components are sensitive to heat e.g., DRAM

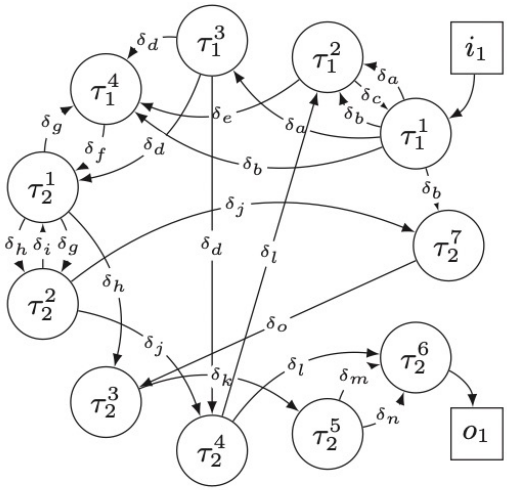
Temperature aware mapping

- Center of chip tends to be hot naturally
  - Heat radiates from the sides
  - Map computationally intensive workloads at egde
  - Depends on the environment
    - Placement of heat sink
    - Position of fan, etc.

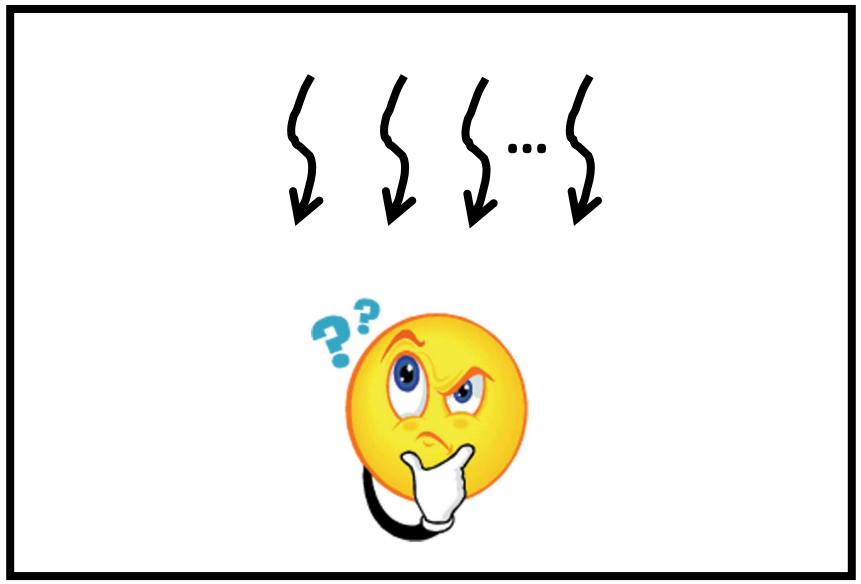


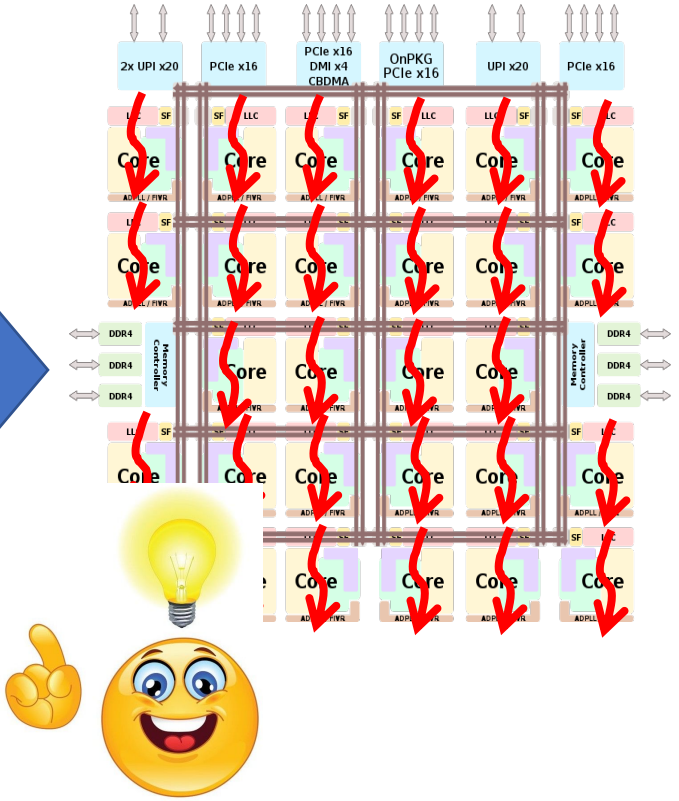
Communication-aware Mapping





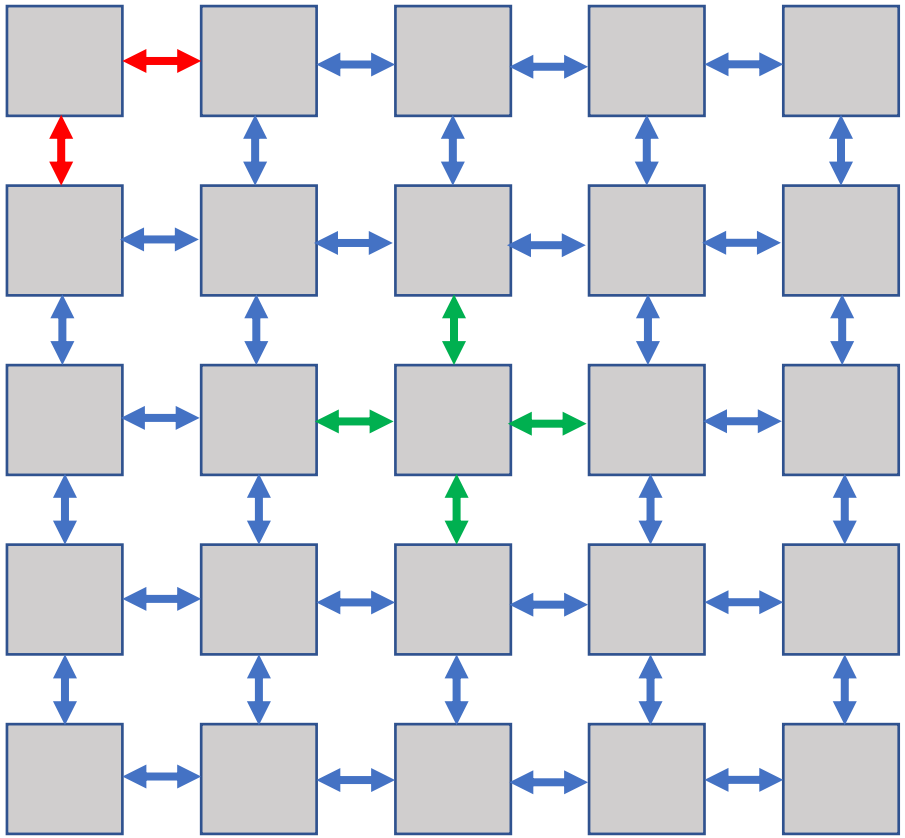
(b) Data exchanges

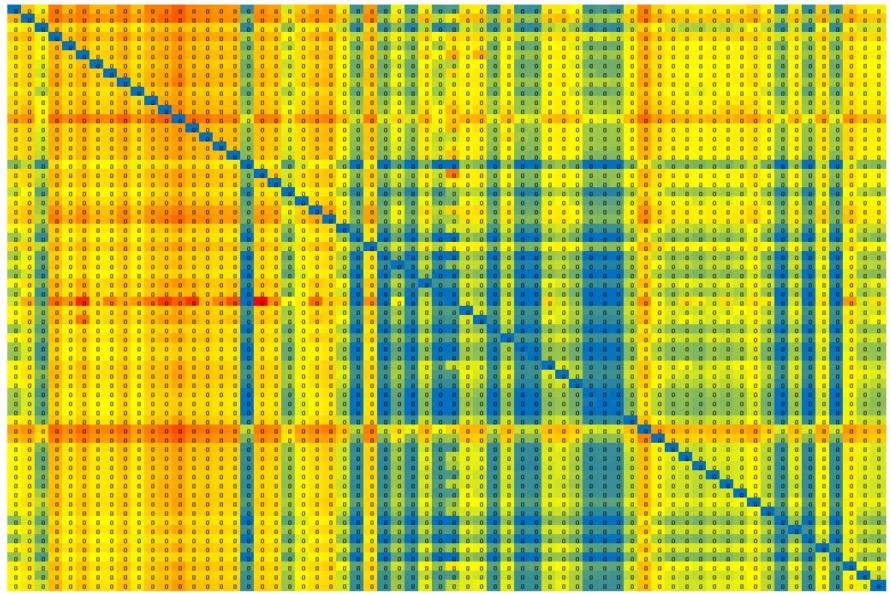




- Given an application behavior, what is the best NoC that we can design?
- Application behavior represented using traffic patterns here

Communication in manycore





- Core-I to Core-j traffic
- Gem5 simulations
- SPLASH2 & PARSEC

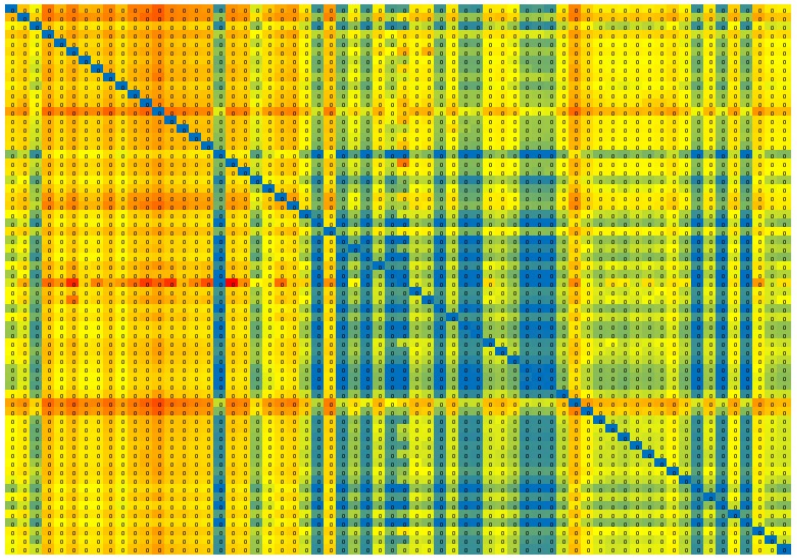
Traffic patterns (1)

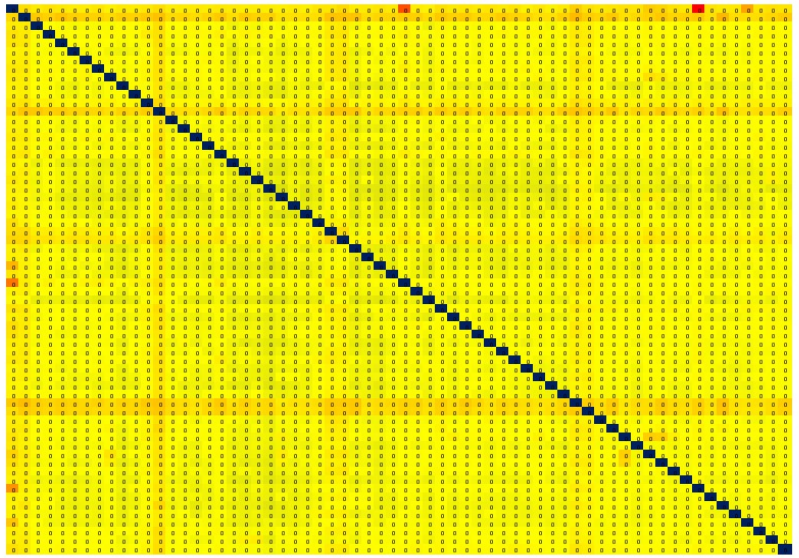
Low

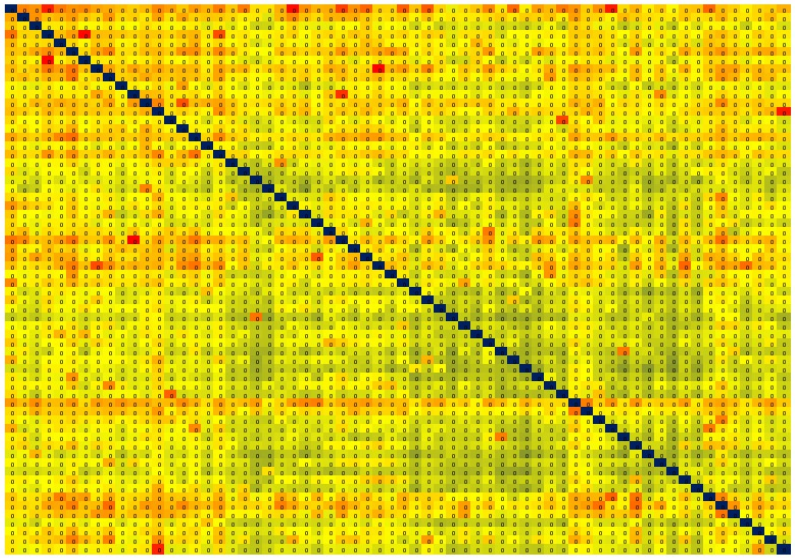
High

Receiver Threads: 1-64

Sender Threads: 1-64







Canneal

- Simulated annealing (SA) to minimize the routing cost in chip design
- **Domain**: Computer engineering

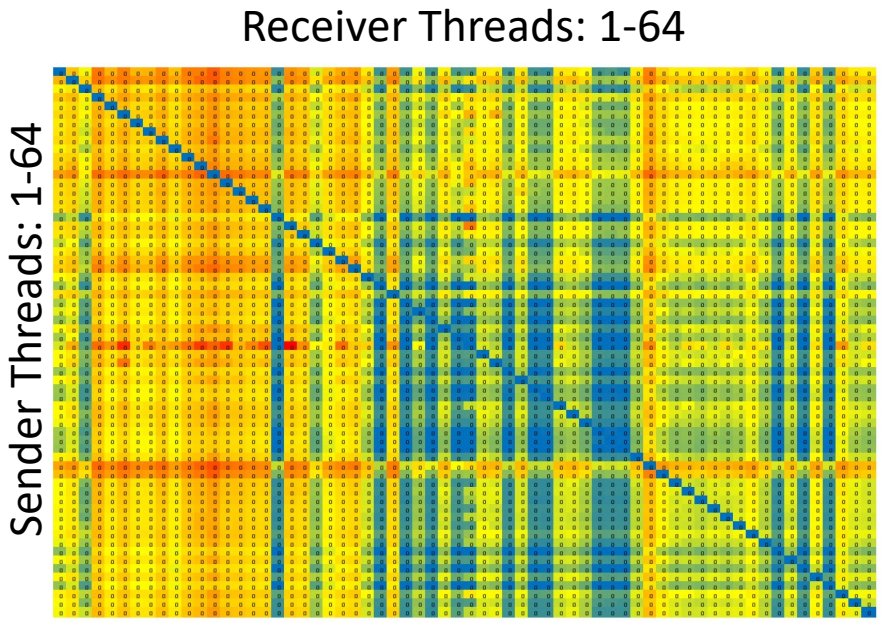
Fast Fourier Transform (FFT)

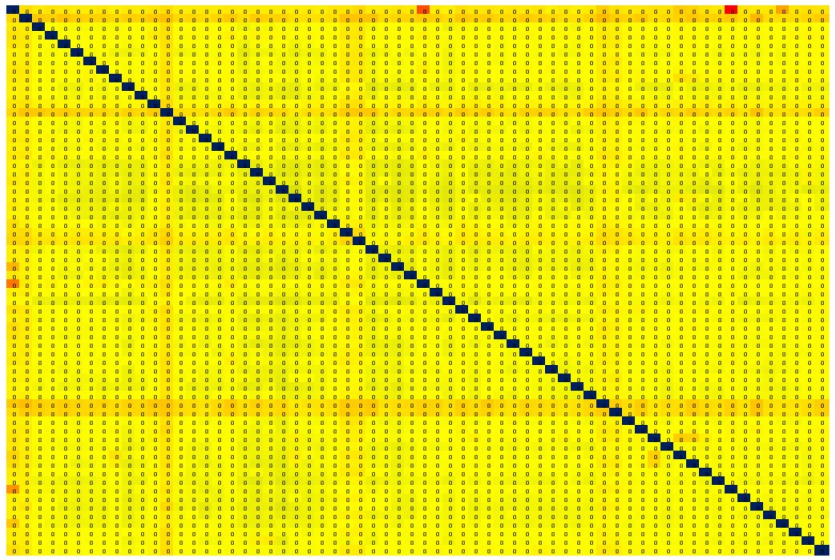
- Computes the fourier transform of a sequence/signal to conert from time domain to frequency domain and vice versa
- **Domain**: Signal processing

Fluidanimate

- Simulate an in- compressible fluid for interactive animation purposes
- **Domain**: Animation

Traffic pattern (2)



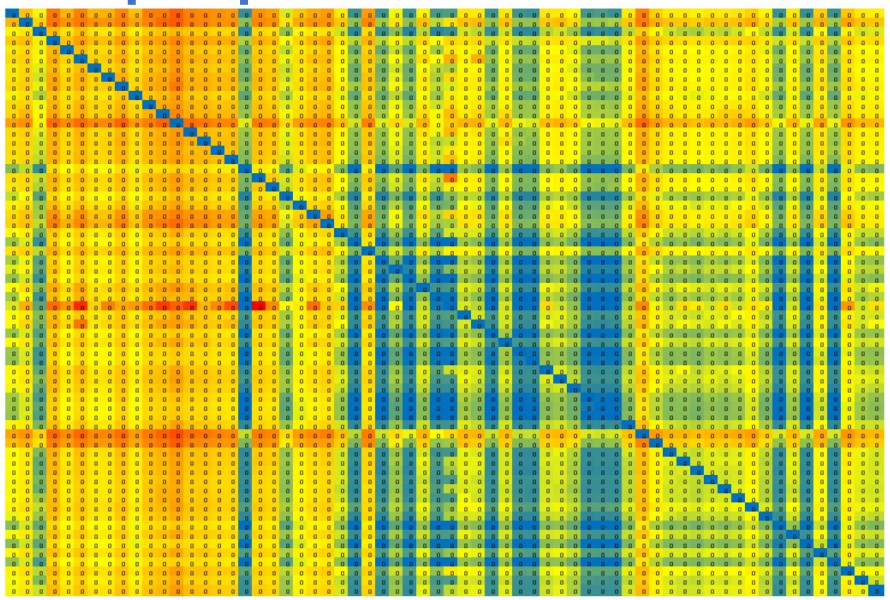


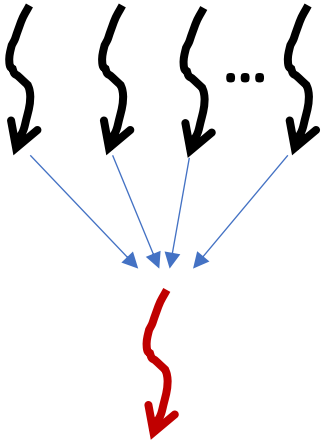
- Canneal
- Unbalanced traffic
- Concentrated between handful of threads only
- Mostly short ranged and some long ranged
- Few-to-few nature

- FFT
- Uniform traffic
  - Both short and long range
  - All-to-all nature

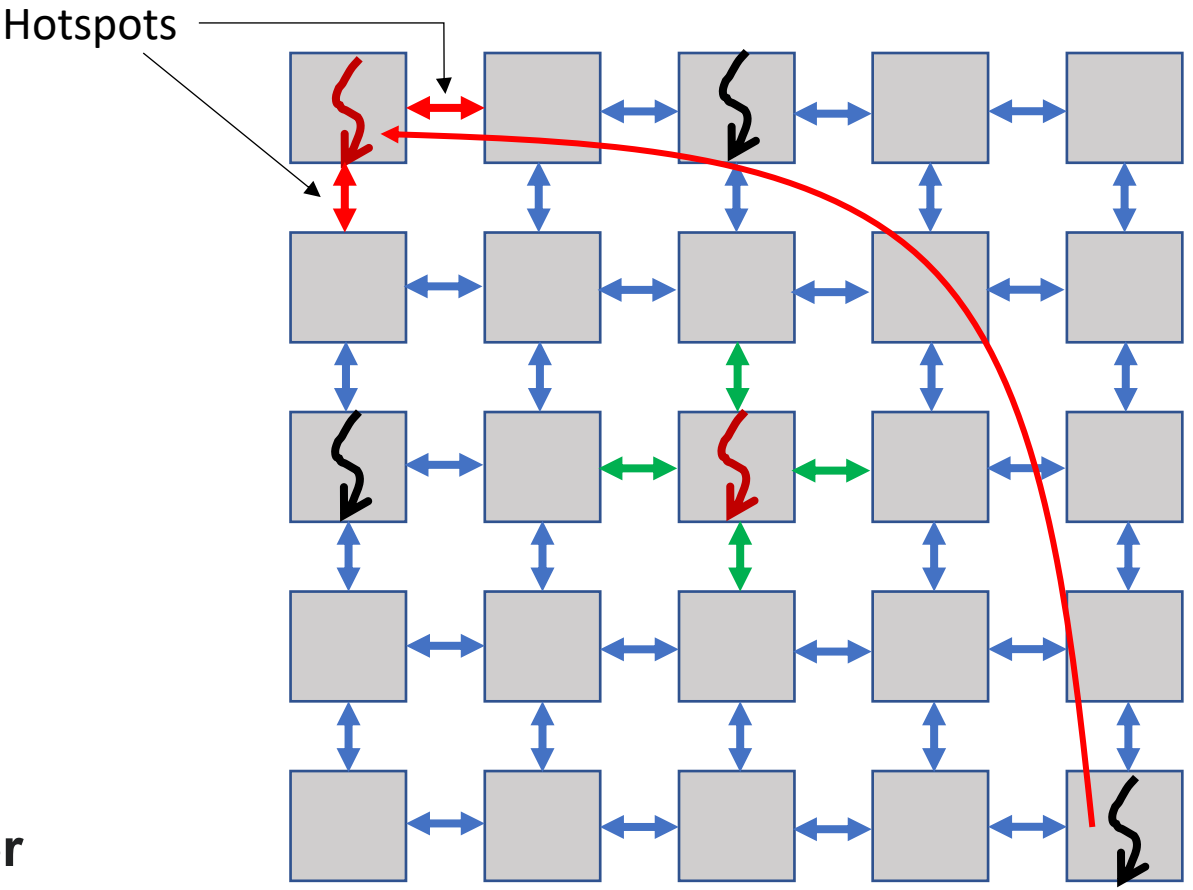
Traffic pattern aware Mapping (1)

Receives data from every thread





- Case-1: Map to corner
  - Only 2 links to reach there
  - Longest path: 8 hops
- Case-2: Map to center
  - 4 links to reach there
  - Longest hop: 4 hops



Longest path: 8 hops

Traffic hotspots

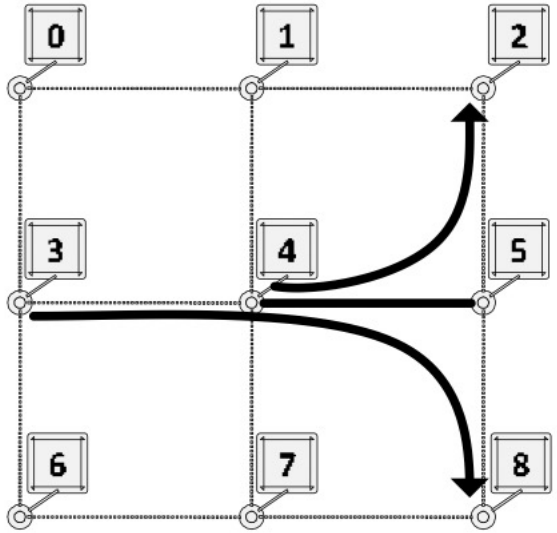


Figure 1. NoC Contention (Config 1)

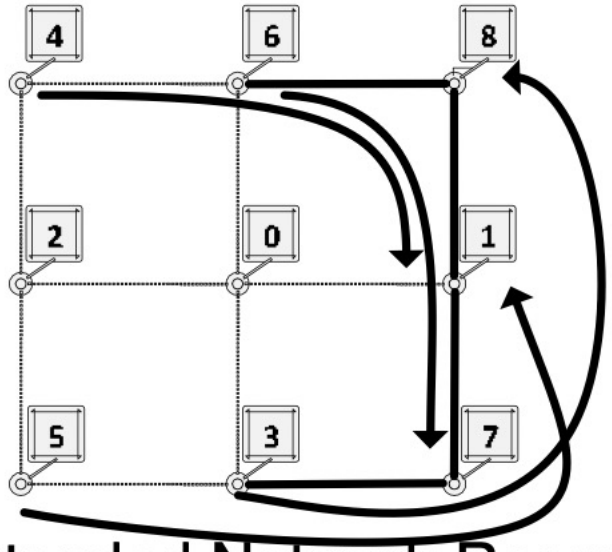


Figure 3. Contended Network Resource (Config 2)

- **Poor mapping contributes to traffic hotspots due to congestion**
- **Mapping should address this**

ML traffic in CPU-GPU system

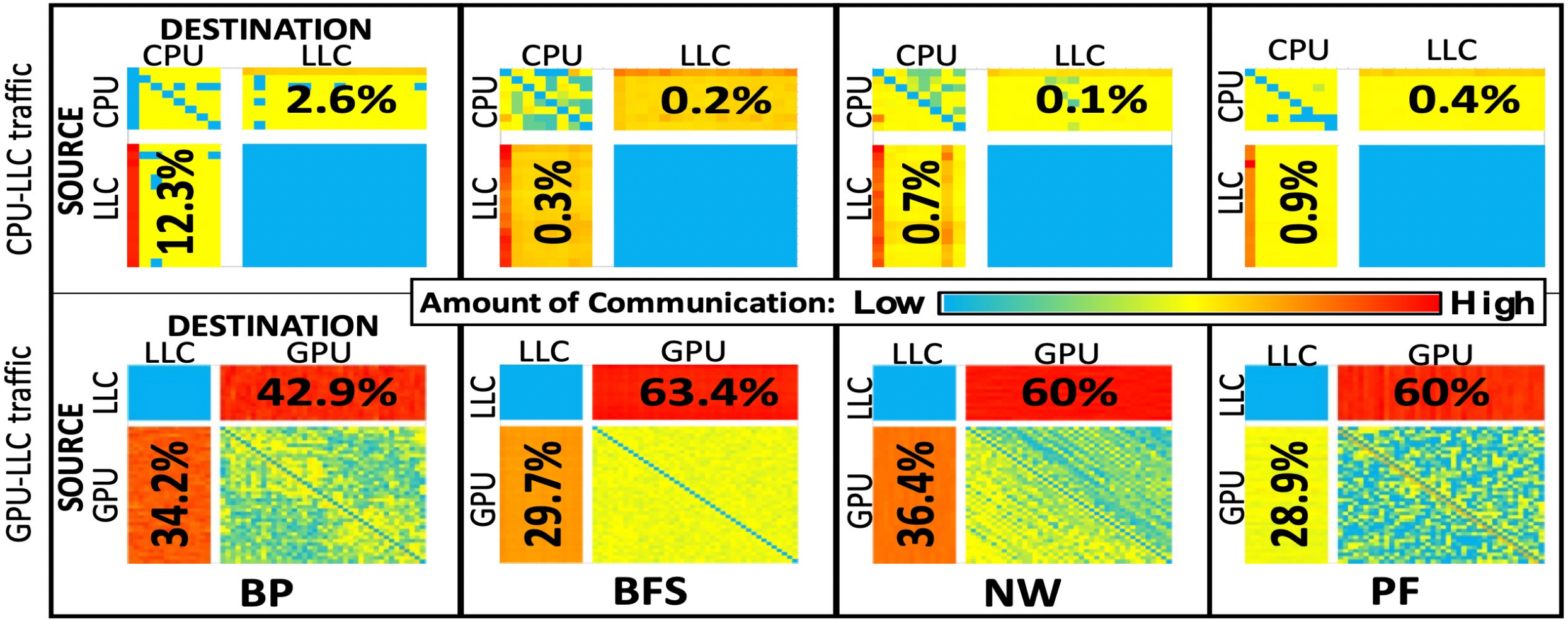


Fig. 1. Traffic pattern heat map for four applications (BP, BFS, NW, and PF) running on a 64-tile heterogeneous manycore system. The numbers indicate percentage of total traffic contributed by that section (*e.g.*, CPU-LLC communication results in 2.6% of total traffic for BP).

Mapping ML to manycore

- Mapping all communication heavy tasks to the center is also a bad idea
- Links on periphery will not be utilized
- Performance bottleneck, aging
- Optimize the mapping

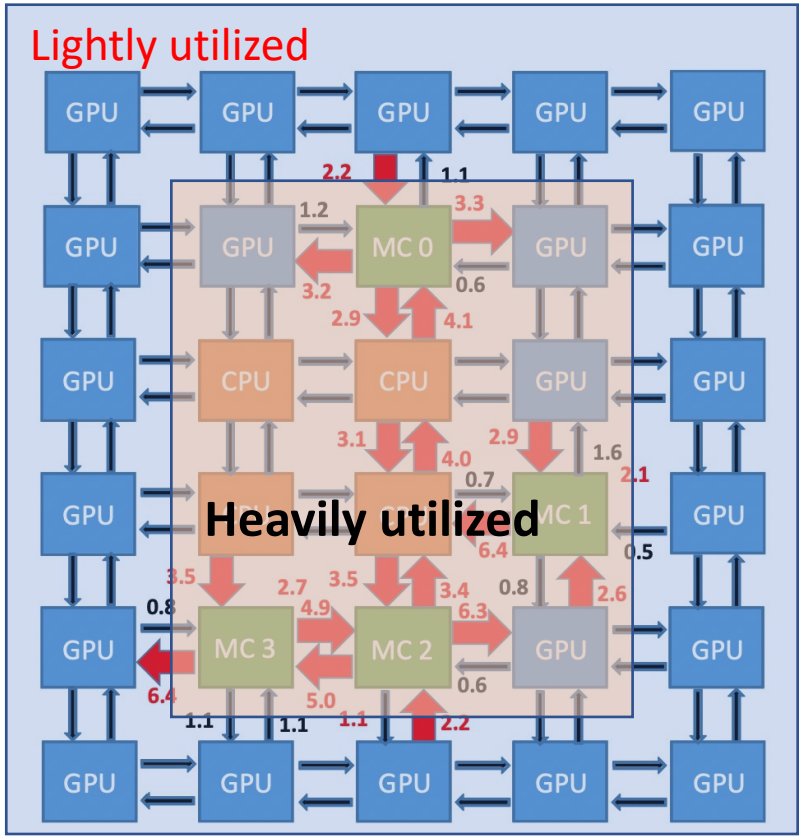


Fig. 8. The central 6×5 portion of the optimized Mesh NoC indicating the link utilizations and the locations of CPUs and MCs for LeNet computations. All link utilizations are normalized with respect to the mean link utilization. The red arrows indicate the bandwidth bottlenecks whose utilization is at least 100% more than the mean.

W. Choi et al., "On-Chip Communication Network for Efficient Training of Deep Convolutional Networks on Heterogeneous Manycore Systems," in IEEE Transactions on Computers, vol. 67, no. 5, pp. 672-686, 1 May 2018, doi: 10.1109/TC.2017.2777863.

Evaluating a mapping/design

- **GPU objective: Maximize throughput**
  - **Throughput (load balancing)**
    - $\overline{U} = \frac{1}{L}\sum_{k=1}^{L}\left(\sum_{i=1}^{R}\sum_{j=1}^{R} f_{ij} \cdot p_{ijk}\right)$ (Average link utilization)
    - $\sigma = \sqrt{\frac{1}{L}\sum_{k=1}^{L}(U_k - \overline{U})^2}$ (Std. Dev. of link utilization)
- **CPU objective: Low latency**
  - **Latency**
    - **Lat =** $\frac{1}{C*M}\sum_{i=1}^{C}\sum_{j=1}^{M}\left(r \cdot h_{ij} + d_{ij}\right) \cdot f_{ij}$ (Zero load latency)
- **3D-specific objective: Temperature**
  - **Max. on-chip temperature**
    - $T = \max_{n,k}\left\{\sum_{i=1}^{k}\left(P_{n,i}(t)\sum_{j=1}^{i} R_j\right) + R_b\sum_{i=1}^{k} P_{n,i}(t)\right\} * T_H$

[1] **B. K. Joardar**, R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu and R. Marculescu, "Learning-Based Application-Agnostic 3D NoC Design for Heterogeneous Manycore Systems," in IEEE Transactions on Computers, vol. 68, no. 6, pp. 852-866, 1 June 2019

Effect of mapping (1)

- Good mapping solutions lower traffic hotspots
- Less congestion = better performance
- Faster ML

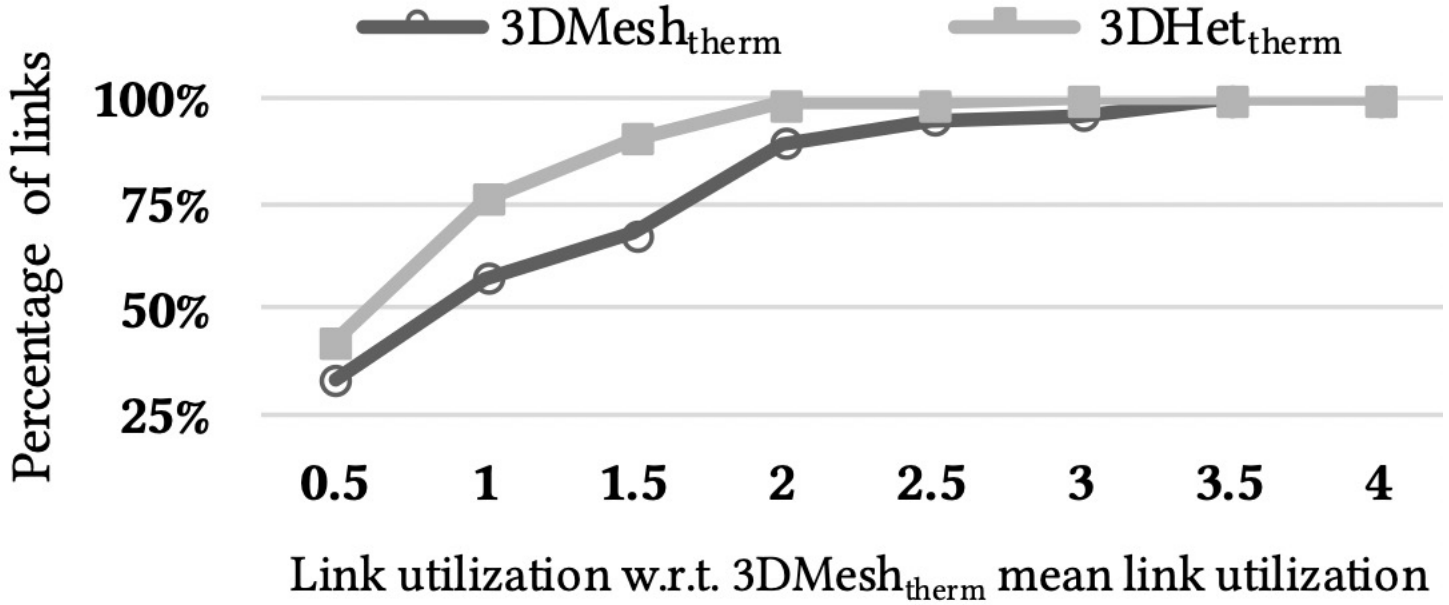


Figure 8: CDF of 3DMesh$_{therm}$ & 3DHet$_{therm}$ link utilizations.

Effect of mapping (2)

- Good mapping solutions lower traffic hotspots
- Less congestion = better performance

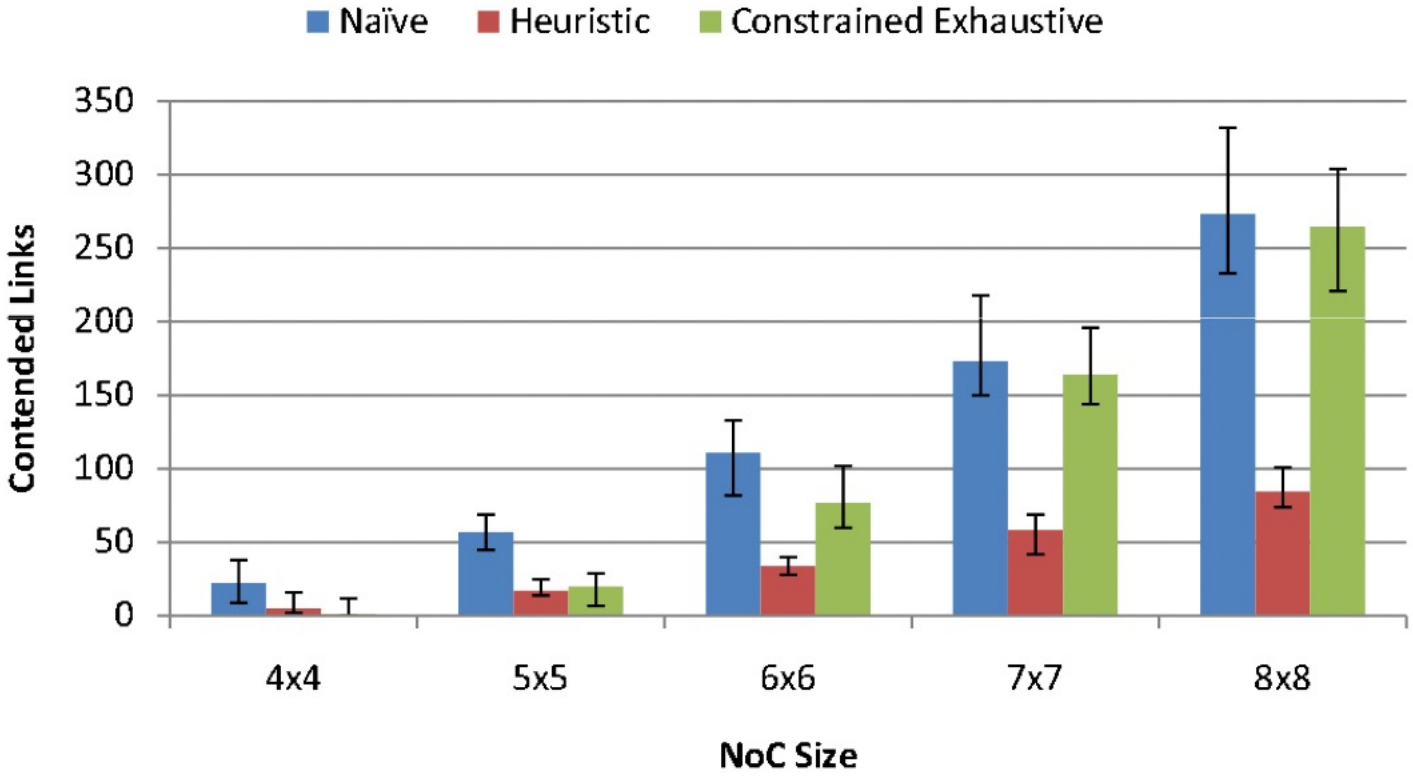
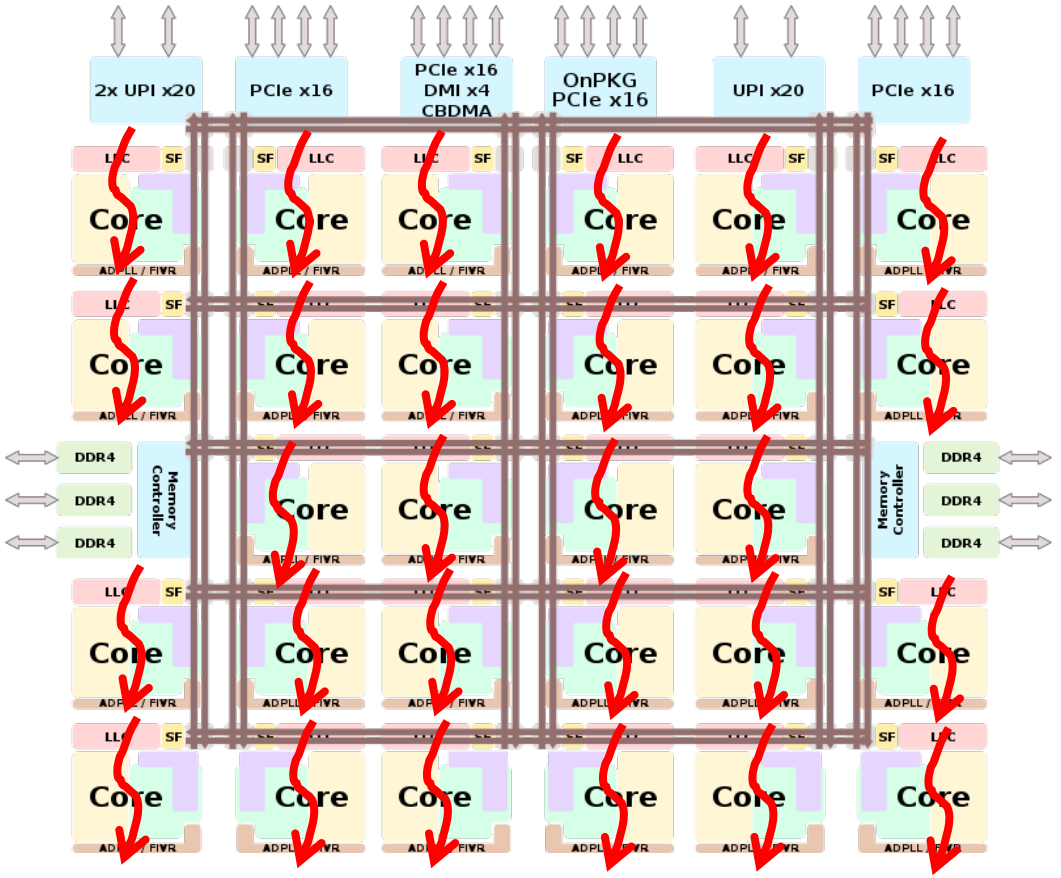


Figure 8. Average Contention for Mapping Strategies

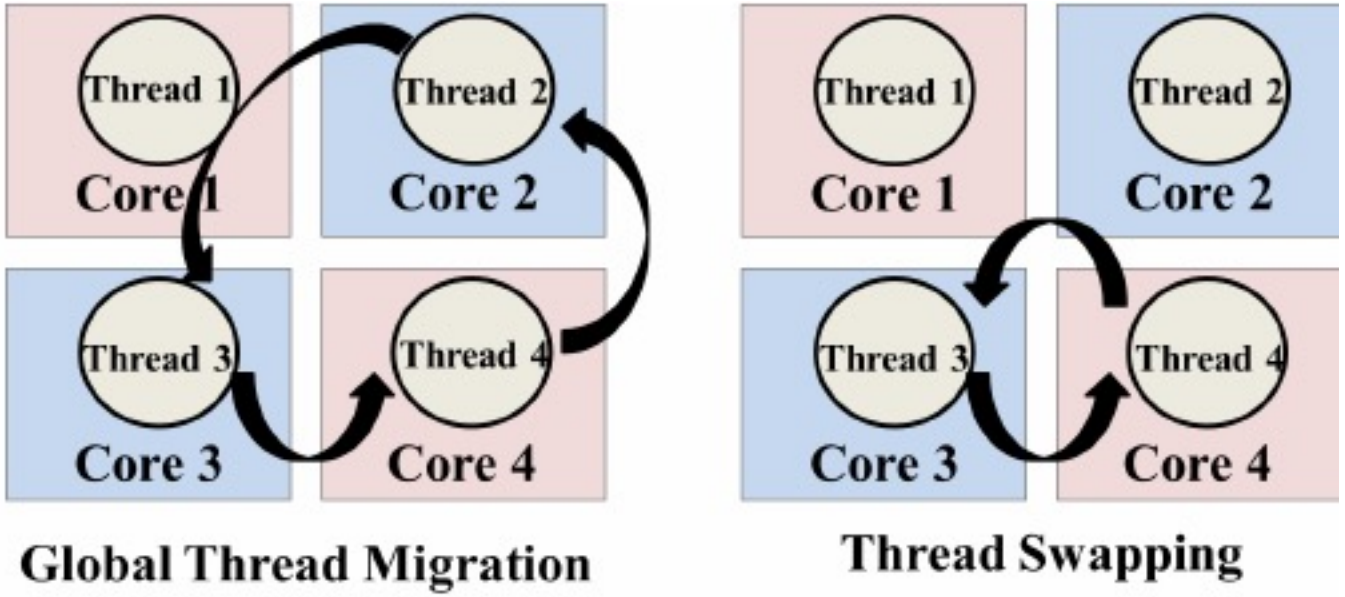
Static vs Dynamic mapping

- So far, mapping was static
- Mapping should be flexible
- Unplanned scenarios may arise during execution
- Change mapping as needed



Thread migration

- Threads often migrate during execution
  - Thermal concerns
  - Power management
  - Load balancing
  - Heterogeneous cores



C. Ravishankar, S. Ananthanarayanan, S. Garg and A. Kennings, "Analysis and evaluation of greedy thread swapping based dynamic power management for MPSoC platforms," Thirteenth International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 2012, pp. 617-624, doi: 10.1109/ISQED.2012.6187557.

How thread migration works?

- Thread migration is complex
- Move everything related to the thread e.g., register, memory contents, PC, etc.
- More communication

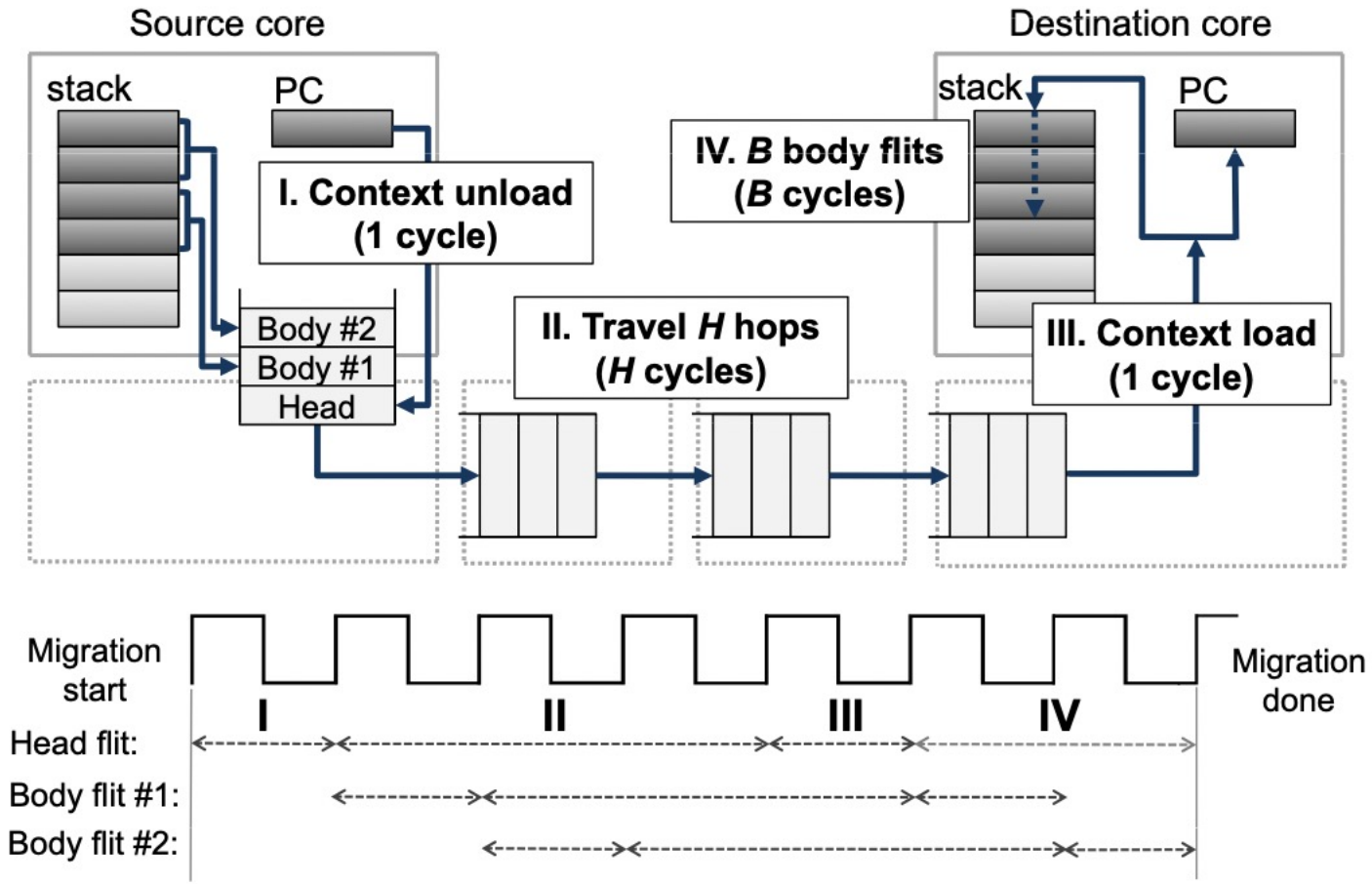
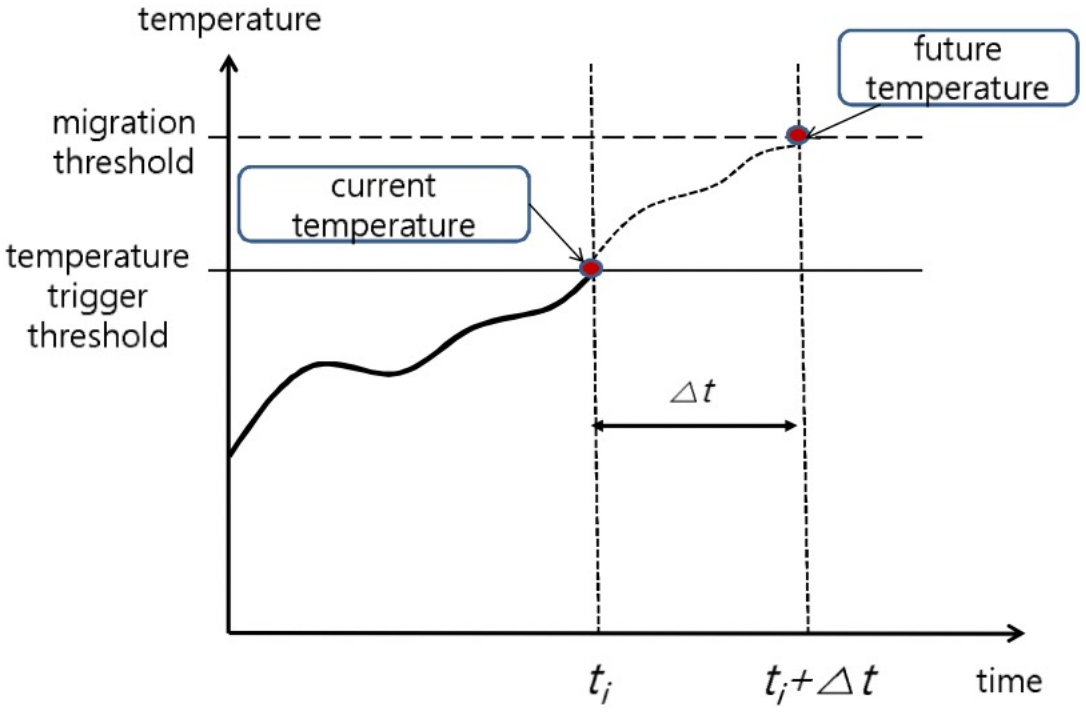
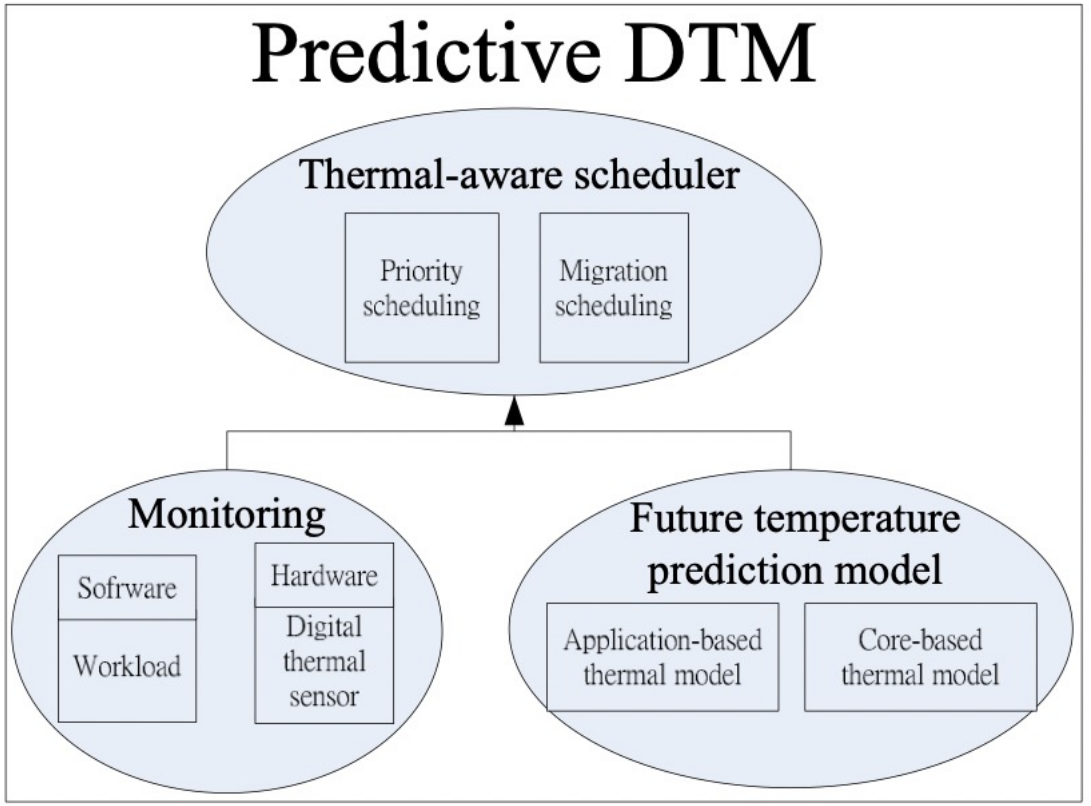


Figure 5: Hardware-level thread migration via the on-chip interconnect. Only the main stack is shown for simplicity.

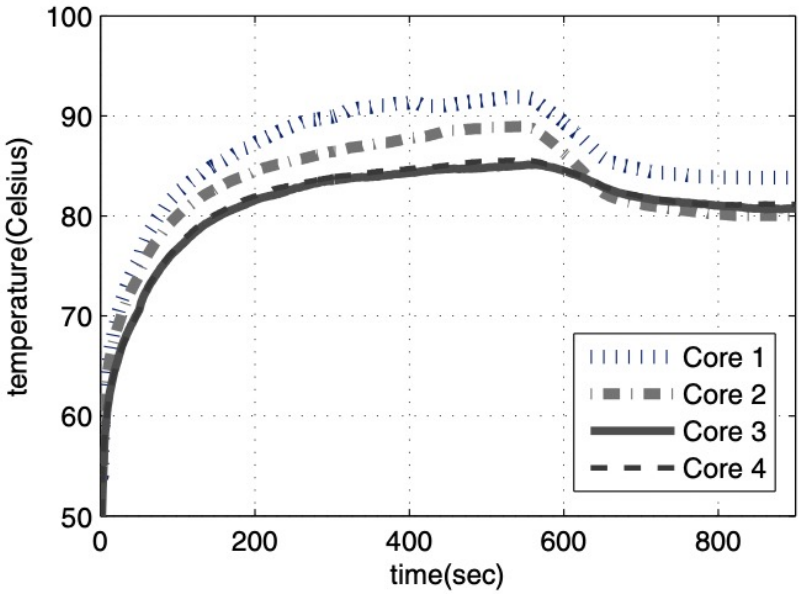
Thermal management using thread migration



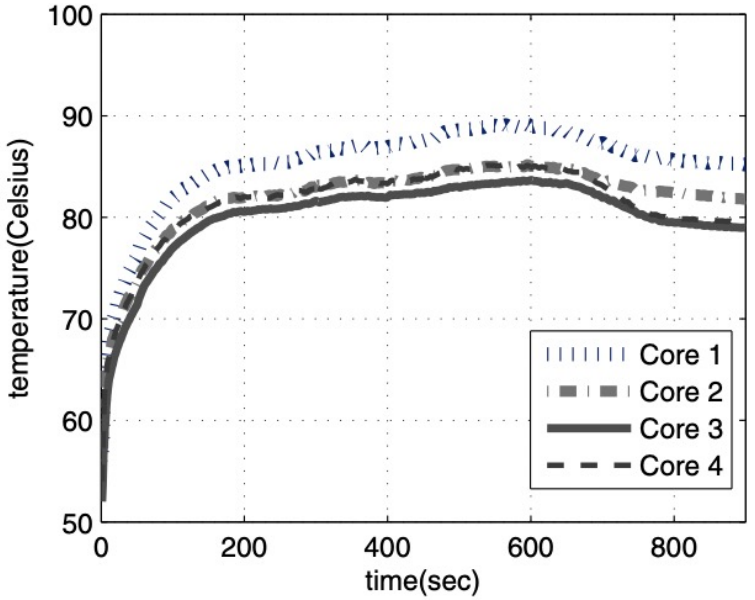


- If temperature increases, performance throttled
- What if we can prevent that from happening?
- Move power-hungry threads to some place else

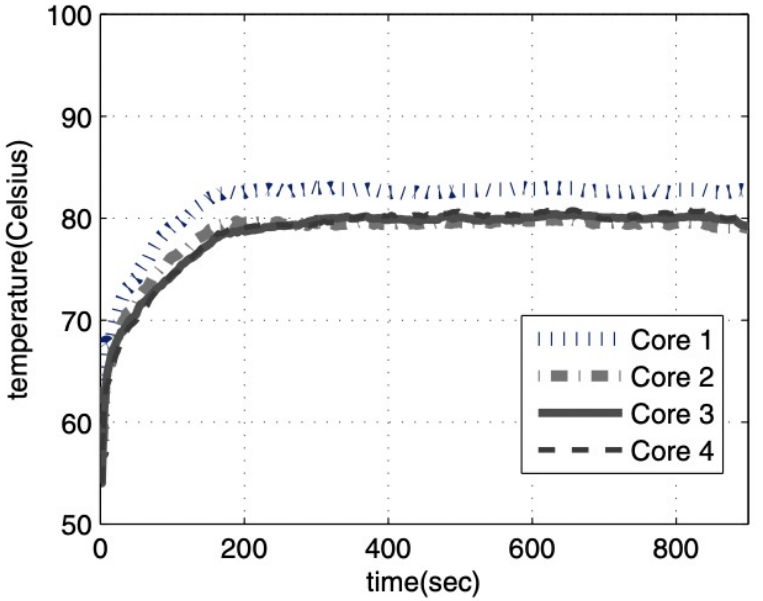
Temperature after thread migration



(a) Without DTM



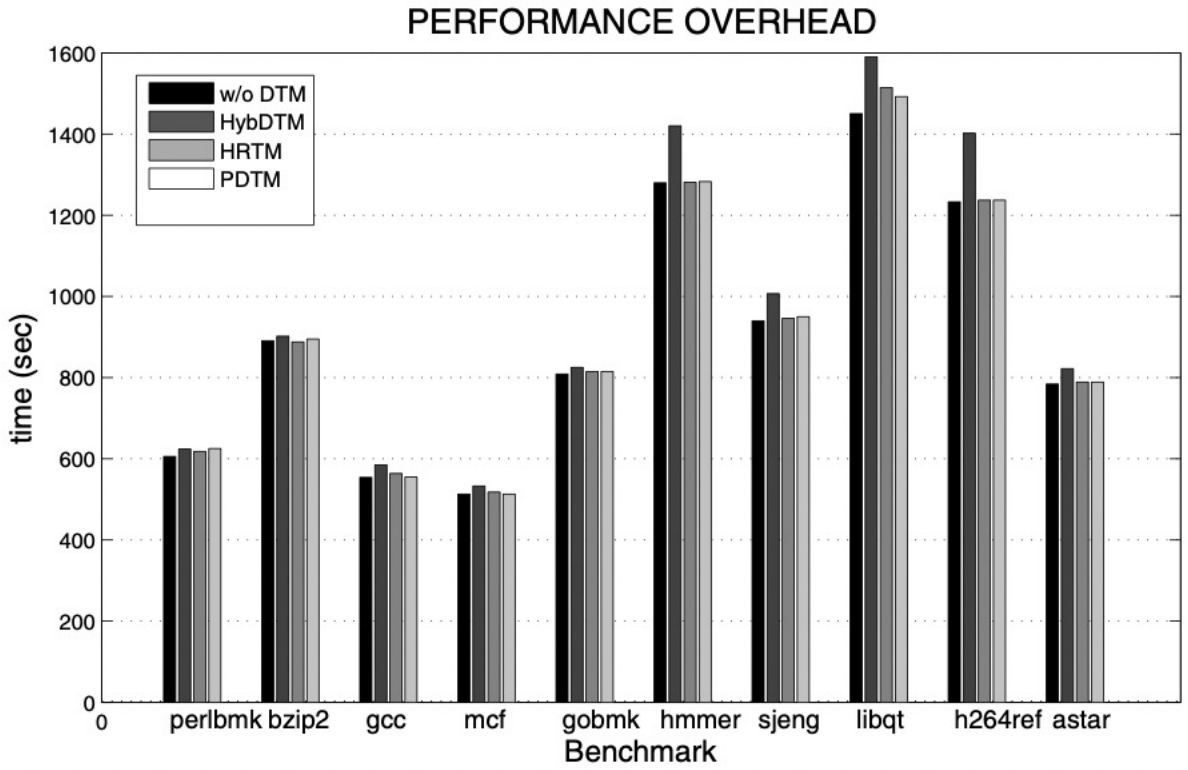
(b) HRTM



(c) PDTM

- **Reduced temperature by ~10C**
- **10C is big deal in ICs**

Performance overhead of thread migration



- Thread migration has performance cost
- ~1% performance overhead
- Summary: No free lunch!