3. Introduction to Heterogeneous Programming

Programming and Architecture of Computing Systems

What is heterogeneous computing?

Syllabus

- Concurrency and Parallelism
- Strong and Weak Scalability
- Quick Overview of Heterogeneous Programming Models

Goals

- Learn the difference between concurrency and parallelism
- Understand message passing and shared memory programming models
- Reason about parallelism scalability, know Amdahl's law

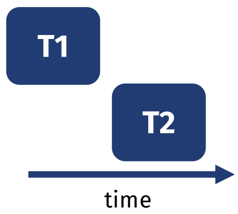
Sequential vs Parallel Execution

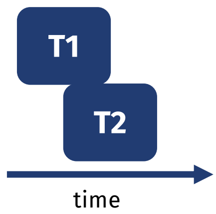
- In **sequential**, we only have **one** stream of instructions
  - Limited by Instruction Level Parallelism

- In **parallel**, **multiple** streams of instructions
  - Limited by the available data or task/function level parallelism

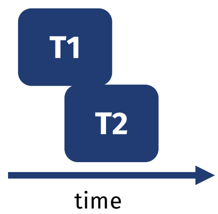
Concurrency vs Parallelism

- Concurrent activities **may** be executed in parallel
  - T1 may overlap with T2





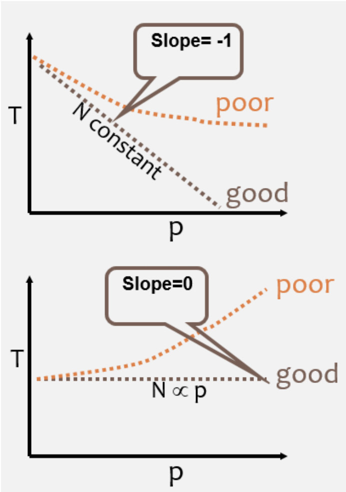
- Parallel activities
  - T2 runs while T1 is running
  - Tend to require more resources



Scalability

- Two types of scaling based on time to solution

- **Strong scaling**:
  - The total program size stays fixed as more cores are added
  - Goal: run the same problem faster
  - Perfect scaling means problem is solved in 1/#P time (#P number of processors/cores)
- **Weak scaling**:
  - The problem size per processor remains fixed as more processors are added
  - Problem size proportional to #P
  - Goal: run larger problem in the same duration
  - Perfect scaling means #P-size problem runs in the same duration as in 1 core



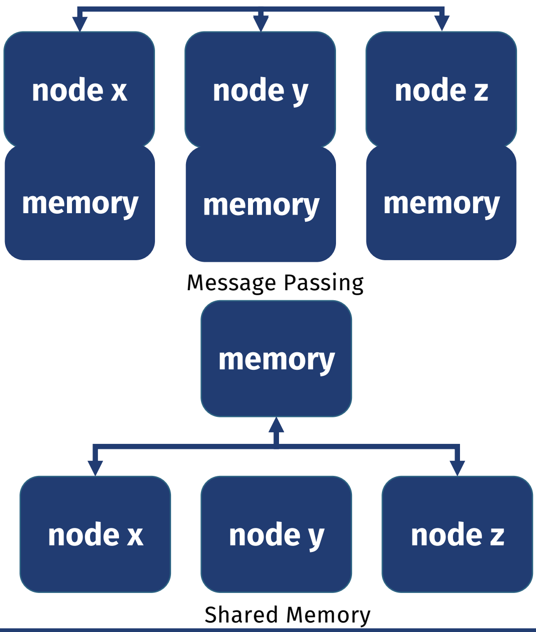
source: https://computing.llnl.gov/tutorials/parallel_comp

Two Main Parallel Programming Models

- Based on how threads communicate and synchronize

- **Message Passing Model** (Distributed Memory):
  - Communication network to connect inter-processor memory
  - Each processor has its own local memory (multiple memory spaces)
  - Harder to program, better memory scalability

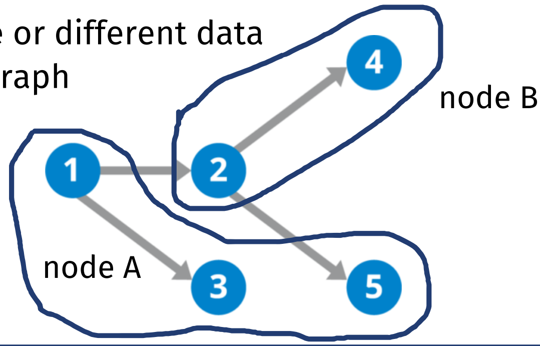
- **Shared Memory**:
  - All processors can access to the same memory and same global address space
  - Easier to program, harder to scale, increasing the number of processors increases the processor-memory bandwidth



Shared Memory Parallelism

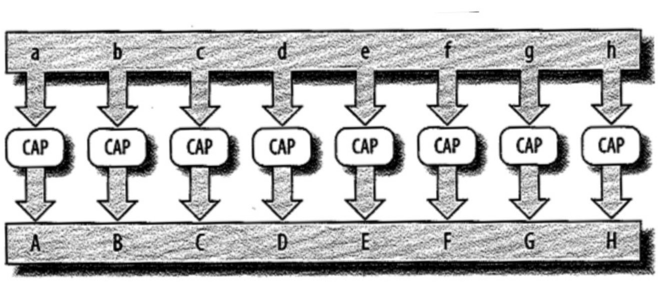
- Data level parallelism
  - Split the data among computing nodes
  - Often associated to regular data structures; e.g., vector

- Task level parallelism
  - Multiple tasks running on the same or different data
  - Think in programs as data acyclic graph



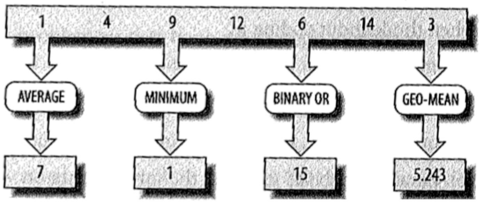
Data Level Parallelism

- Run the **same task**/code on **different data** in parallel
- Example: Capitalize an array
  - Division of data between the tasks that can run in parallel
  - Observation: no dependencies between the data that can cause incorrect results



Task Level Parallelism

- Several functions/tasks on the same data:
  - Example: The Average, minimum, binary or geometric mean tasks can run in parallel



- Several functions/tasks on different data:
  - Computer Vision pipeline

Amdahl's Law

- Potential application speed-up (S) is defined by the fraction of parallelized code (P):

$$S = \frac{1}{1-P}$$

- Introducing the number of cores (C) performing the parallel fraction (P) and the serial fraction (s), the equation can be modeled by:

$$S = \frac{1}{\frac{P}{C} + s}$$

Gene Amdahl

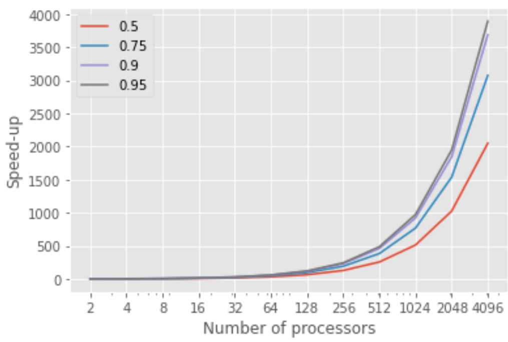
Limits to the scalability

	speed-up			
C	P =.5	P = .9	P=.95	P=.99
10	1.82	5.26	6.89	9.17
100	1.98	9.17	16.80	50.25
1000	1.99	9.91	19.62	90.99
10000	1.99	9.91	19.96	99.02
100000	1.99	9.99	19.99	99.90

Gustafson's Law

- Amdahl's law main assumption: fixed problem size
- Remember weak scaling, what if problem size changes with the number of cores
- Solve a large problem in the same amount of time
- Speed-up (S) is proportional to the number of cores minus the time spent on the serial part (1-P)

$$S = C + (1 - N)(1 - P)$$



John Gustafson

Exercise

- Please visit https://colab.research.google.com/drive/1HbUzXlo5tJ8FQn1Qo5_kM2dfgYGxk17d?hl=es and modify the parallel fraction
- When we code, what should be the percentage of parallel code?

Existing Heterogeneous Programming Models

- Many alternatives for Heterogeneous/Parallel Programming
  - OpenMP for accelerators
  - OpenACC
  - CUDA
  - OpenCL
  - OmpSs
  - StarPu
  - SYCL

OpenMP for accelerators

- Standard for shared memory parallel programming
  - From version 4.0, support for GPUs, …
- OpenMP is based on directives (pragmas)
- OpenMP device model:
  - Host plus same-type accelerators
- Example directive:
  - ```
    // transfer control and data from the host to the device
    #pragma omp target [data]
    …
    ```


OpenMP Heterogeneous Example

```
#pragma omp target data device(0) map(alloc:tmp[:N]) map(to:input[:N)) map(from:res)
  {
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    update_input_array_on_the_host(input);

#pragma omp target update device(0) to(input[:N])

#pragma omp target device(0)
#pragma omp parallel for reduction(+:res)
    for (i=0; i<N; i++)
      res += final_computation(input[i], tmp[i], i)
  }
```

host target host target host

Source: https://www.scc.kit.edu/downloads/scs/Heterogeneous%20Programming%20with%20OpenMP%204..pdf
```

OpenACC

- Designed by Cray, CAPS, Nvidia and PGI for supercomputers
- Compiler directive based
- Host device model
- Goals: Performance and Portability
- Example:

Manage Data Movement

Initiate Parallel Execution

Optimize Loop Mappings

```
#pragma acc data copyin(a,b) copyout(c)
{
  ...
  #pragma acc parallel
  {
  #pragma acc loop gang vector
      for (i = 0; i < n; ++i) {
          c[i] = a[i] + b[i];
          ...
      }
  }
  ...
}
```

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, Manycore

Source: https://www.openacc.org/sites/default/files/inline-files/OpenACC_Course_Oct2018/OpenACC%20Course%202018%20Week%201.pdf

OpenCC Goals Description

Incremental

- Maintain existing sequential code
- Add annotations to expose parallelism
- After verifying correctness, annotate more of the code

Single Source

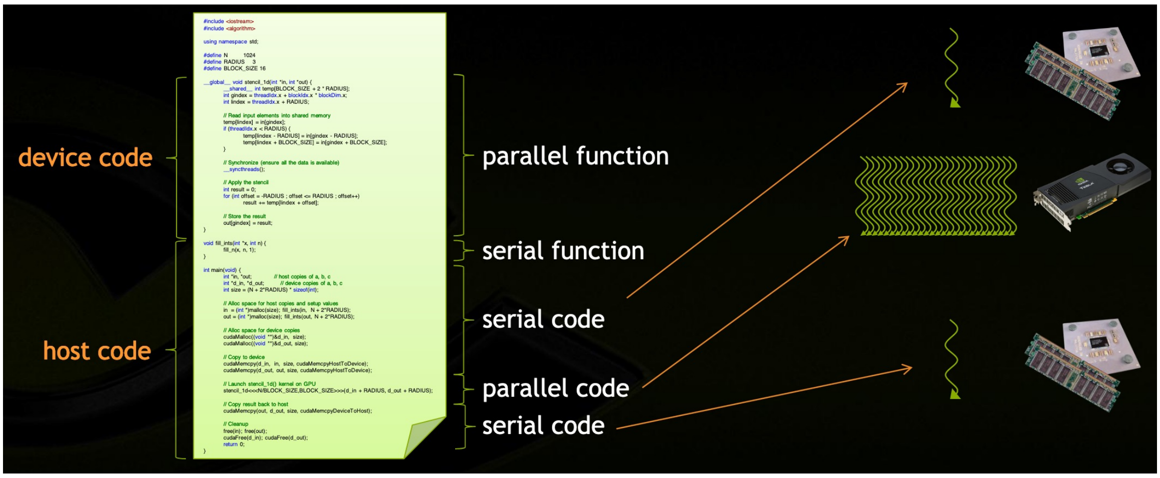
- Rebuild the same code on multiple architectures
- Compiler determines how to parallelize for the desired machine
- Sequential code is maintained

Low Learning Curve

- OpenACC is meant to be easy to use, and easy to learn
- Programmer remains in familiar C, C++, or Fortran
- No reason to learn low-level details of the hardware.

CUDA

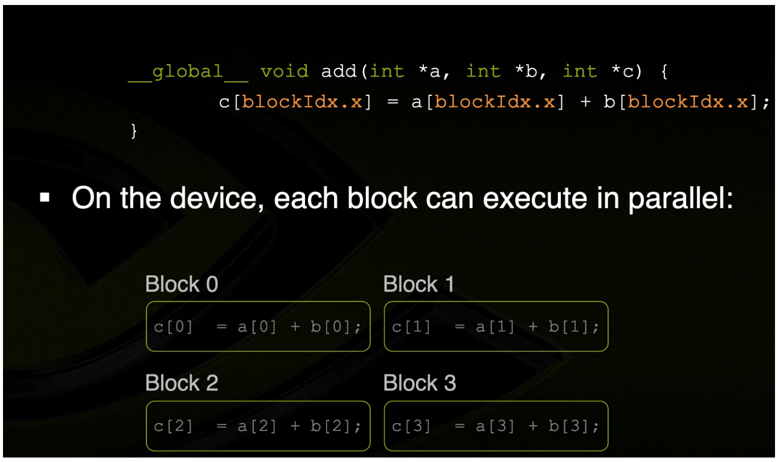
- Developed by NVIDIA for their GPUs
- Host device model
- Large set of optimized libraries (computer vision, machine learning, math, ...)



source: https://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf

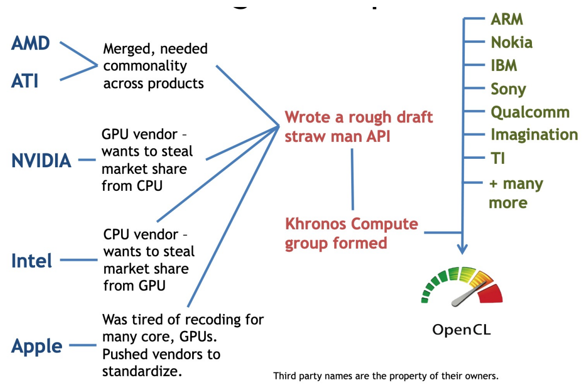
CUDA

- Works with grid of ranges
- Express the operation for a single element of the range



OpenCL

- Royalty-free multiple device standard
- Host-device model
- Precursor of SYCL



source: *https://www.nersc.gov/assets/pubs_presos/MattsonTutorialSC14.pdf*

OpenCL example kernel

- Replace loops with functions (a kernel) executing at each point in a problem domain
  - E.g., process a 1024x1024 image with one kernel invocation per pixel or 1024x1024=1,048,576 kernel executions

Traditional loops

```
void
mul(const  int n,
    const  float *a,
    const  float *b,
           float *c)
{
  int i;
  for (i = 0; i < n; i++)
    c[i] = a[i] * b[i];
}
```

OpenCL

```
__kernel void
mul(__global const float *a,
    __global const float *b,
    __global       float *c)
{
  int id = get_global_id(0);
  c[id]  = a[id] * b[id];
}
// execute over n work-items
```

OmpSs and StarPu

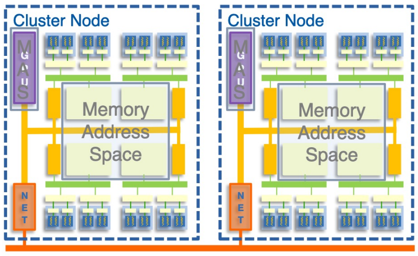
- Academic proposals
- OmpSs from BSC/UPC has influenced OpenMP (directive based)
- StarPu from Inria
- OmpSs provides support for multiple models (CUDA, OpenCL, ...)

```
int A[SIZE];

#pragma omp target device (smp) \
   copy_out([SIZE] A)
#pragma omp task
   matrix_initialization(A);

#pragma omp taskwait

#pragma omp target device (cuda) \
   copy_inout([SIZE]A)
#pragma omp task
{
   cu_matrix_inc<<<Size,1>>>(A);
}
```



source: https://materials.prace-ri.eu/327/4/OmpSsQuickOverviewXT.pdf

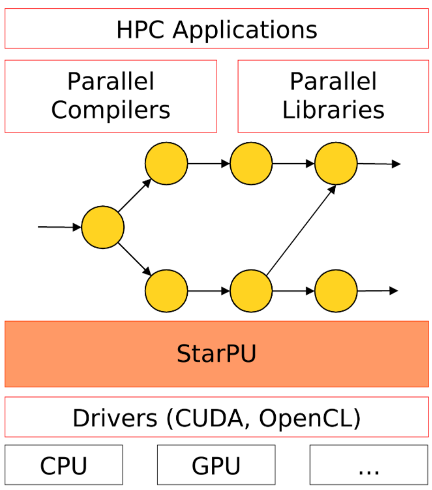
StarPu Runtime System

94

The StarPU runtime system

The need for runtime systems

- “do dynamically what can’t be done statically anymore”
- Compilers and libraries generate (graphs of) tasks
  - Additional information is welcome!
- StarPU provides
  - Task scheduling
  - Memory management



source: https://starpu.gitlabpages.inria.fr/tutorials/2019-05-EXA2PRO/StarPU_introduction.pdf

Conclusions

- Ensure you understand the optimization opportunities (scaling, parallel fraction, …) of your program before start optimizing
  - “premature optimization is the root of all evil” Donald Knuth

source: nytimes.com

- The application will guide the parallelism model
- Many existing Heterogeneous Programming Models