PERFORMANCE ENGINEERING

Lecture 3: Performance Modeling

April 21th, 2022

Ana Lucia Varbanescu
a.l.varbanescu@uva.nl
(room C3.109)

Today ...

- Recap Roofline
- Modeling as part of PE
- Analytical modeling
  - Basic laws (recap)
  - First steps into analytical models
- Examples & discussion

Roofline in practice

- Calculate Roofline model for hardware
- Calculate AI for application
- Measure performance
  - Calculate GFLOPs
- Draw graph and analyse results
  - Check if HW models are correct
  - Check if results make sense
- Propose optimizations to address the observed bottleneck

In general, we have three optimization strategies:
- Improve AI
- Improve memory performance
- Improve compute performance

Other Roofline developments

- CARM – Cache-Aware Roofline Model
- Instruction-based Roofline Model
- Roofline for FPGAs
- Energy roofline
- …

https://ideas-productivity.org/events/hpc-best-practices-webinars/#webinar010
https://crd.lbl.gov/divisions/amcr/computer-science-amcr/par/research/roofline/publications/

Today ...

- ~~Recap Roofline~~

- Modeling as part of PE

- Analytical modeling
  - Basic laws (recap)
  - First steps into analytical models

- Examples & discussion

PE's systematic approach

- A 5-stages process, partially iterative

1. Understand requirements
2. Understand current performance
3. Can it be done?
4. How can it be done?
5. Tuning

   Not there yet? => back to 2
6. Analyze the result

What have we addressed so far?

1. Understand requirements

So far …

- Best possible performance
  - Efficiency/utilization of peak
- Speed-up vs. original/reference
- Minimum performance
  - ”real-time” or otherwise
- …

Important

- Metrics of success are application-specific
  - And usually derived from measurable quantities

2. Current performance

So far …

- Execution time
- GFLOPs or GB/s

Important tools

- Profilers
- Performance events/counters
  - Raw
    - E.g. : memory LD/ST, L1 accesses, number of FMAs, number of ADDs,…
  - Derived
    - E.g. : hit ratio’s, GFLOPs, …

3. Can it be done?

So far …

- Roofline model
- Anything else?

e.g., Amdahl's law?

Important

- Machine characterization
  - Based on simplified model!
- Application characterization
  - Based on simplified model
    - Arithmetic intensity
    - Operational intensity
    - Instruction intensity

PE’s systematic approach

- A 5-stages process, partially iterative

1. Understand requirements
2. Understand current performance
3. Can it be done?
4. How can it be done?
5. Tuning

   Not there yet? => back to 2
6. Analyze the result

Performance models!

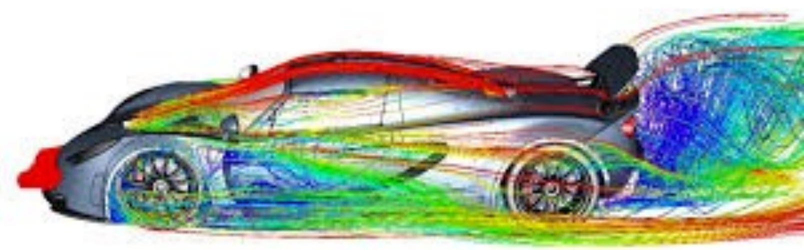
PERFORMANCE MODELING

An overview

Informally …

- What is a model … ?

- A simplification of “real-life” that allows closer study/analysis;
- Models are used for different things
  - Behavior-analysis
  - Simulation
  - Prediction



Informally …

- Performance model (in this course)…
  - Any form of model that enables us to study/understand/predict performance for computing systems & applications

- Performance models “answer” performance questions
  - What is the peak performance of a system?
  - What is the max performance of the application?
  - What if I use acceleration?
  - What is the performance of the app for an unseen input?
  - What is the range of performance of an app?
  - When is algorithm1 faster than algorithm2?
  - What is the energy efficiency of this application?

Performance modeling

Goal: “say something” about the performance of an *application* on a *machine*

- application != code
- machine != hardware

Same 4(+) steps:

- Step 1: Choose modeling strategy
- Step 2: Build model
- Step 2’: Calibrate model
- Step 3: Validate & assess model
- Step 4: Apply model

Performance modeling (cont'd)

Many different modelling strategies:

- Benchmarking (time consuming & hard to analyse)
- Data-driven/"ML"-based (time consuming, little insight)
- Simulation (time consuming)
  - Cycle-accurate
  - Event-based
- Analytical (difficult to build)

Most models remain manually build, calibrated, or tuned.
Automated performance modeling = holy grail for many domains

Performance model assessment

- Applicability
  - How applicable is my model ?
- Accuracy
  - How accurate is my model ?
- Effort
  - How much effort does it take to build by model?
    - Calibration included
  - How much effort does it take to use the model?
- Resources
  - Modeling: How many resources I need to build my model?
  - Prediction: How many resources I need to use my model?

Benchmarking

- **Goals**: "how will my application execute on this hardware?"
- **Idea**: find a representative application to measure and generalize findings

- Requirements:
  - Code
    - Representative applications
  - Hardware
    - Representative machines
  - Data
    - Representative workloads
- Effort
  - Measurement time (high)
- Resources
  - Modeling: Machine(s) must be available (high)
  - Prediction: some form of extrapolation (low)

Data-driven/ML-based modeling*

- **Goals**: “can I predict the performance of my application for any input on my hardware”
- **Idea**: use past execution to build a statistical model of the application performance

- Requirements:
  - Application
    - The application itself
  - Hardware
    - The machine itself or a representative one
  - Data
    - MANY representative workloads
- Effort
  - Data collection on the target machine (high)
  - Apply ML tools (moderate/low)
- Resources
  - Modeling: Real machine(s) (high)
  - Prediction: Apply model on new input (low)

*Also called “statistical” in the sense that the ML used is related on statistics.

Simulation

- **Goals**: “can I predict the performance of my system?”
  - System = application + data + hardware
- **Idea**: simulate the operation of the system and measure different events

- Requirements:
  - Application
    - A model of the application or the code itself
  - Hardware
    - A simulator of the machine (high-level)
  - Data
    - A representative workload
- Effort
  - Build the simulator (high)
  - Use the simulator (moderate/low)
- Resources
  - Modeling: None.
  - Prediction: Machine to run simulation (moderate)

Analytical modeling

- **Goals**: “can I predict the performance of my system/application?”
  - System = application + data + hardware
- **Idea**: model the operation of the system in a symbolic model and calibrate for real system

- Requirements:
  - Application
    - A model of the application or the code itself
  - Hardware
    - A high level model of the hardware
  - Data
    - A model of a representative workload
- Effort
  - Build the model (high)
- Resources
  - Modeling: None.
  - Prediction: None

ANALYTICAL MODELING

Examples & strategies

Fundamental idea

- Machine model = simplified *functional* model of the machine, including …
  - Execution model (+ costs)
  - Memory model (+ costs)
  - Communication model (+ costs)

- Application model = simplified application *functionality*, in terms of operations, like …
  - A sequence of kernels/phases
  - A sequence of instructions
  - …

- Performance = cost estimation of executing the app operations on the hardware

Performance bound example: Amdahl's law

What is the max performance the app can achieve when parallelized/accelerated?

Amdahl's Law

- Application :
  - *s* = sequential work
  - (1-*s*) = parallelizable work
  - *p* = number of processors
  - *S* = application speedup

$$
\begin{aligned}
S &= T_{seq}/T_{par} \\
&= 1/(s + (1-s)/p) \\
&\leq 1/s
\end{aligned}
$$

- It assumes that the parallelization is perfect
  - (1-*s*)/*p* assumes 100% efficiency => that is, linear speed-up
- It fixes the problem size
  - *s* can be* problem-size-dependent

*and often is…

Amdahl's Law as analytical model

- Machine model ?
  - Sequential unit and parallel units (*p*-wide)
    - Execution for parallel units: T/p
  - No memory or instruction-level model
- Application model ?
  - A series of kernels, some parallelizable
- Cost estimation (execution time)?
  - T = Tseq + Tpar/p
- Is this an accurate model?
- How would you validate it?

Another example: The Roofline model?

How much performance can be achieved?

Roofline model prediction

- Attainable GFlops/sec
  = min(Peak Floating-Point Performance,
  Peak Memory Bandwidth * Operational Intensity)

Compute intensive

Memory intensive

- Provides an upper bound for a given kernel.
- Can this help with Amdahl's law?

Performance bounds models

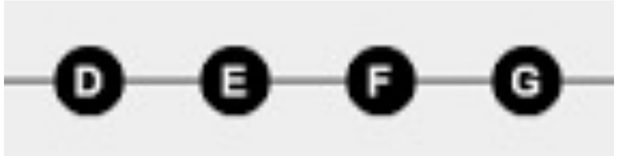
- Amdahl’s law
  - “The speed-up of a parallel application is limited by its sequential part.”
    - We can extend that to any optimization besides parallelism
  - **Answers the question: is the parallelization/optimization worth the effort?**

- Roofline model
  - Estimates realistic performance bounds a “kernel” can achieve
    - It takes the application into account through its AI
    - It takes the platform into account through its throughput and bandwidth
  - **Answers the question: how much can I realistically optimize a kernel?**

Performance bounds models

- Take an application:

$T = T_D + T_E + T_F \ldots$



Assume …

T_D = reading/writing files => NOT optimized

T_E , T_F = to be run in parallel on **p** processors

= to be accelerated by factors $\mathbf{f_E}$, $\mathbf{f_F}$

- Amdahl:
  - $T_E' = T_E/\mathbf{f_E}$, $T_F' = T_F/\mathbf{f_F}$
    - Assumes $\mathbf{f_E} = \mathbf{f_F} = p$
  - $S_A = T/(T_D+T_E'+T_F') < T/T_D$
- Roofline: realistic estimates for T'_E, T'_F (or f_E, f_F, or S_E, S_F)
  - P = min(peak_FLOPs, AI * peak_BW) => Ti'' = #ops / Pi
  - $S_R = T/(T_D+T_E''+T_F'')$
- In general: $S_R < S_A$

Roofline model as analytical model?

- Machine model ?
- Application model ?
- Cost estimation ?
- Is this an accurate model?
- How would you validate it?

Roofline model as analytical model?

- Machine model ?
  - A bunch of cores and DRAM
  - Maybe some memory hierarchy
- Application model ?
  - Kernels/loops
  - Arithmetic / operational intensity
- Cost estimation ?
  - Peak performance in GFLOPs/s

- Is this an accurate model?

- How would you validate it?

Building analytical models

What is the application performance for a given machine?

Assumptions

- Simplified hardware models
  - RAM – Random Access Machine for sequential code
    - Sequential, in-order architecture
  - PRAM – Parallel RAM
    - Multiple cores, homogeneous
    - Shared memory
    - Add-on: memory hierarchy
- Application = a collection of instructions
  - Arithmetic operations
  - Memory operations

MAIN IDEA: express the application in primitive operations of the machine.
NOTE: finer granularity => more complex model, easier calibration.

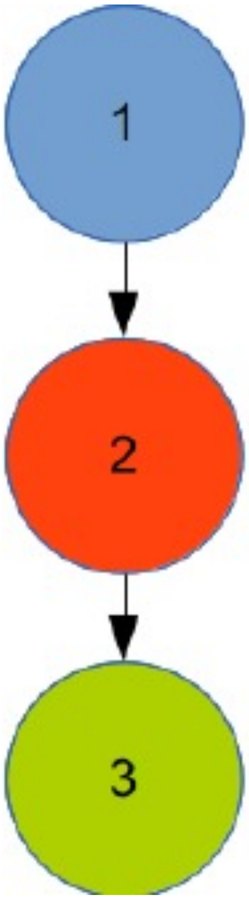
A simplistic model [1]

$$T = T1 + T2 + T3$$

$$Ti = T\_comp + T\_comm$$

- Are we done ?
  - Measure
    - Calibrate now
  - Model
    - Finer-grain modeling => keep modeling => calibrate later
    - Calibrate later => easier calibration

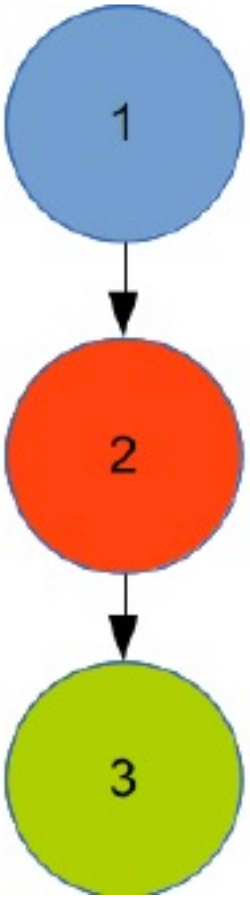
Key idea: Find the right balance between modeling and calibration, which depends on goal and accuracy requirements.



A simplistic model [2]

Ti = T_comp + T_comm

- T_comp
  - Count number of operations
  - Multiply by time_per_operation
- T_comm
  - Define communication
    - Memory operations?
    - Inter-process communication?
  - Compute data items communicated
  - Multiply by time per element/batch



A more "elaborate" model

T = T_comp + T_mem + T_comm + T_par

- Serial : Ts
  - T_comp = N_alu_ops * t_op
  - T_mem = N_mem_ops * t_mem
  - T_comm = 0
  - T_par = 0
- Parallel : Tp
  - T_comp = N_alu_ops/**p** * t_op
  - T_mem = N_mem_ops/**p** * t_mem
  - T_comm = N_comm * t_comm
  - T_par = T_overhead

Overhead due to parallelism (mix of comp & mem)

Communication/synchronization (e.g., atomics, barriers, inter-node comm, etc.)

Refers to memory operations.

Another (analytical) model here …

Not the only possible model!

Tp = N_alu_ops/p * t_op + N_mem_ops/p * t_mem +…

A more "elaborate" model

T = T_comp + T_mem + T_comm + T_par

- Serial : Ts
  - T_comp = N_alu_ops * t_op
  - T_mem = N_mem_ops * t_mem
  - T_comm = 0
  - T_par = 0

Overhead due to parallelism (mix of comp & mem)

Communication/synchronization (e.g., atomics, barriers, inter-node comm, etc.)

There is no strict rule about which operations go to which "term" in the model (that is, you can still see this as T_comp + T_mem).

However, separation to illustrate the different effects of parallelism/distribution is recommended.

- T_par = T_overhead

Not the only possible model!

Tp = N_alu_ops/p * t_op + N_mem_ops/p * t_mem +…

Improving the model ?

1. Take into account different arithmetic operations
2. Take into account different memory operations
3. Take into account the memory hierarchy
4. Take into account the overlapping of compute and load//store operations (memory operations)
5. Take into account the overlapping of computation and communication

$$T = \overbrace{\text{N\_alu\_ops}/p * \text{t\_op}}^{1} \overset{4}{+} \overbrace{\text{N\_mem\_ops}/p * \text{t\_mem}}^{2,3} \overset{5}{+} \ldots$$

Model different operations (1,2)

- Arithmetic operations of different kinds have different latency and throughput
  - Add, shift, …
  - Multiply, divide
  - SIMD or not
  - Int, float, double
  - …
- Memory operations have different latency and throughput
  - Load
  - Store
  - Cache hierarchy & policies

Model caches

- In general, two options
  1. Separate the accesses in cache levels and use different access times
  2. Use a statistical approach to determine an average access time

- Advantages? Disadvantages?

Cache Performance Metrics

- Miss Rate
  - Fraction of memory references not found in cache (misses / accesses) = 1 – hit rate
  - Typical numbers (in percentages):
    - 3-10% for L1
    - can be quite small (e.g., < 1%) for L2, depending on size, etc.
- Hit Time
  - Time to deliver a line in the cache to the processor (~ cache latency)
    - includes time to determine whether the line is in the cache
  - Typical numbers:
    - 4 clock cycle for L1
    - 10 clock cycles for L2
- Miss Time
  - Time required to service a miss (~ cache latency + miss penalty)
    - typically 50-200 cycles for main memory (Trend: increasing!)
  - Care must be taken to compute it, depending on given metrics *and* potential misses in multiple layers.

Average access time

T_access = hit_rate * hit_time + (1-hit_rate)*miss_time

- Compare 99% hits vs. 97%
  - Consider:
    cache hit time of 1 cycle,
    cache miss time of 100 cycles
  - Average access time:
    97% hits: 0.97 * 1 cycle + 0.03 * 100 cycles = **4 cycles**
    99% hits: 0.99 * 1 cycle + 0.01 * 100 cycles = **2 cycles**

How can we obtain hit/miss ratio?

- Using measurements
  - Use LIKWID
  - Use perf
  - … any other tool

- Using yet another analytical model ☺
  - Knowledge of the architecture
    - Cache model, policies, line sizes …
  - Knowledge of the application/compiler
    - What goes into cache & when

Which is faster? (12p)

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
   sum = 0.0;
   for (k=0; k<n; k++)
     sum += a[i][k] * b[k][j];
   c[i][j] = sum;
 }
}
```

```
for (k=0; k<n; k++) {
 for (i=0; i<n; i++) {
  r = a[i][k];
  for (j=0; j<n; j++)
   c[i][j] += r * b[k][j];
 }
}
```

```
for (j=0; j<n; j++) {
 for (k=0; k<n; k++) {
   r = b[k][j];
   for (i=0; i<n; i++)
    c[i][j] += a[i][k] * r;
 }
}
```

Model the cache behavior of these examples and pick the best one!

Assume:

- L1 cache only
- Data type = double
- Cache line = 32B = 4 x double
- N is very large
  - Cache cannot fit multiple rows
- Replacement: LRU

Back to our performance model …

$$Tp = \overbrace{N\_alu\_ops1/p * t\_op1 + N\_alu\_ops2/p * t\_op2 + \ldots +}^{\textit{T\_comp, on p processors, assume perfect parallelism}}$$

$$\underbrace{+\ldots\ N\_mem\_ops/p * t\_mem}_{\textit{T\_mem, on p processors}} + \ldots$$

No comp/mem overlap

Other terms if needed (comm, overhead)

$$t\_mem = hit\_rate * t\_hit + (1\text{-}hit\_rate) * t\_miss$$

Might change with different p values!

- This is a symbolic performance model
  - May allow for performance ranking!
- For an actual prediction … ?
  - Determine N_op's
  - Determine t's

The topic of our next lecture(s).

Determining number of operations

- By hand:
  - At algorithmic level => number of useful operations
  - Exclude "overhead" => loop index computation, if statements, …
  - Pro's: easy to separate
  - Con's: if overhead is high, accuracy drops

- Profilers:
  - Determine number of operations of each kind
  - Pro's: accurate
  - Con's: difficult to separate core from overhead

Determining the latency per operation

- Theoretical latency
  - From catalogues, in cycles
- Microbenchmarking
  - Isolate operations and measure

An example: histogram

- Assume an application computing the histogram of an image.
  - Image size = W x H
  - Bins = B
- Algorithm:

```
for pixel = 1 .. WxH
    bins[IMAGE[pixel]]++
```

An example: histogram

T = WxH x (T_iteration)

T_iteration = T_compute + T_memory

Compute ops: ++ => T_compute = t(++)

Memory ops: (RD1+RD2+WR2)

RD1 : read from IMAGE

missRate: 1 in (size_cache_line)/(sizeof(pixel)

RD2/WR2 : read/write from BINS (cached)

Hit rate: 100%

T_memory = 2 x t_L1 + (1-missRate) x t_L1 +

+ (missRate) x t_L2

T_iteration = t(++) + 2 x t_L1 + (1-mR) x t_L1 + mR x t_L2

Models of parallel computation

So far …

- Single-kernel applications
  - Matrix multiplication
  - Histogram

- Parallelism = data-parallelism
  - Each processor = same computation + part of the data

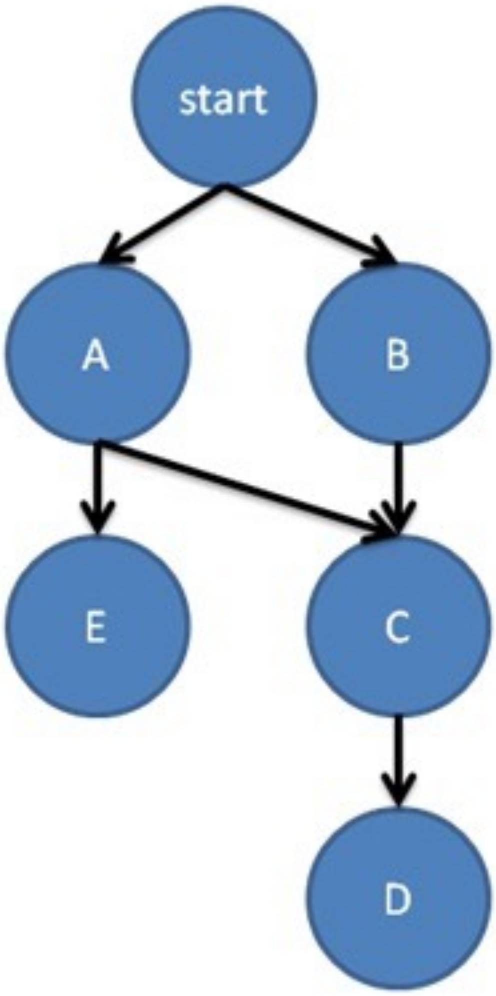
- What about larger applications and their parallelism?

At application level

- Application = DAG of tasks/operations
  - Operations = nodes
  - Dependencies = edges
    - Dependency = child-after-parent
- Two metrics:
  - The amount of **work** to be executed

    T1 = A+B+C+D+E
  - The **span** of the application
    - Also called critical-path length or depth

    T∞ = B+C+D
- Parallelism: T1/T∞
  - the average amount of work per step.

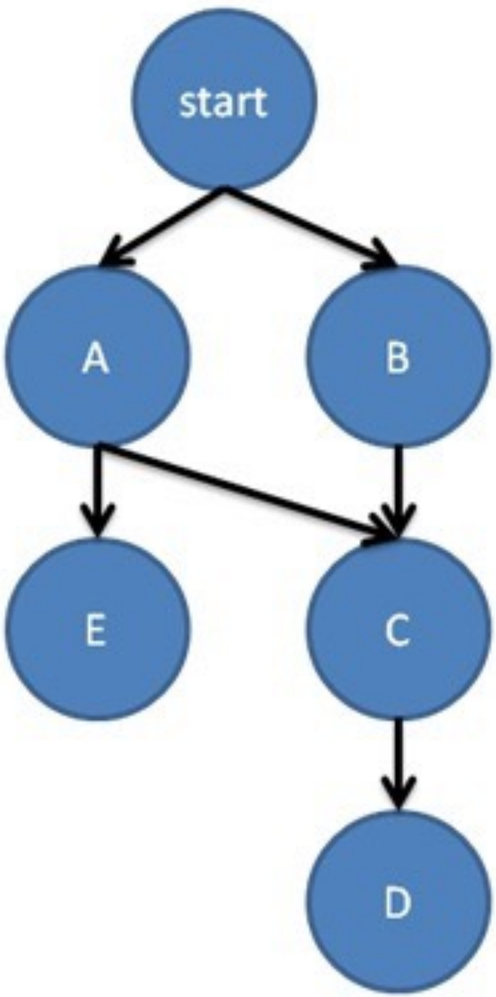


At application level

- Tp = time using p processors
  - Cost = p*Tp
- Two laws for parallelism & performance
  - Work law: p*Tp >= T1 (the cost is at least the work)
  - Span law: Tp >= T∞ (p processors cannot outperform an infinity of processors)

- Performance bounds: the performance of parallel processing on p>1 units is bounded by the work and span of the application.

- Actual performance
  - Resources constraints + Scheduling
  - If p >> parallelism => low efficiency !



Practice with work & span

- Sequential vs. Parallel (A -> B vs. A || B)
  - Work?
  - Span?
- Reduction ?
  - Work?
  - Span?
- Atomic update
  - Work?
  - Span?

Practice with work & span

- Sequential vs. Parallel
  - Work?
    - Seq: A+B Par: A+B
  - Span?
    - Seq: A+B Par: max(A,B)

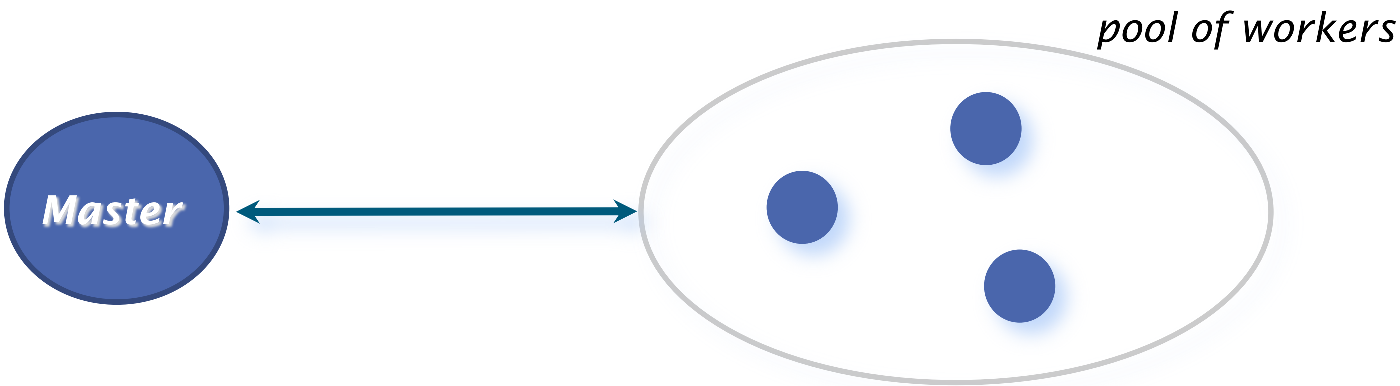
- Reduction ?
  - Work: O (N)
  - Span: O(N) or O(log N)

- Atomic update ?
  - Work: P x T_op
  - Span:
    - Worst case: P x T_op
    - Best estimate: #contention x T_op

Models of Parallel Computation

- For parallel programming we use two-level models of computation:
  - Conceptual level : defining tasks and data interactions
    - farmer/worker
    - divide and conquer
    - data parallelism
    - function parallelism
    - bulk synchronous
  - System level : implementation
    - (logical) data spaces and programs
    - communication and synchronization

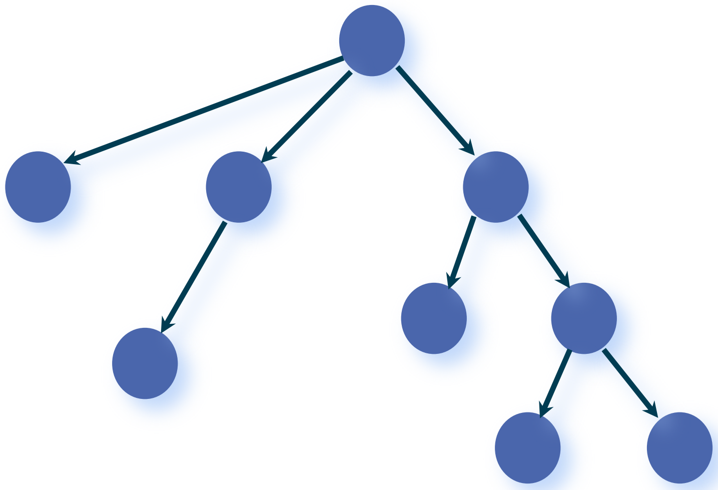
Master/worker model



- A pool of identical worker processes is formed,
- The farmer manages the work for the workers
- When finished, a worker gets another job.
- Pull model: workers ask for work
- Push model: farmer gives work to workers

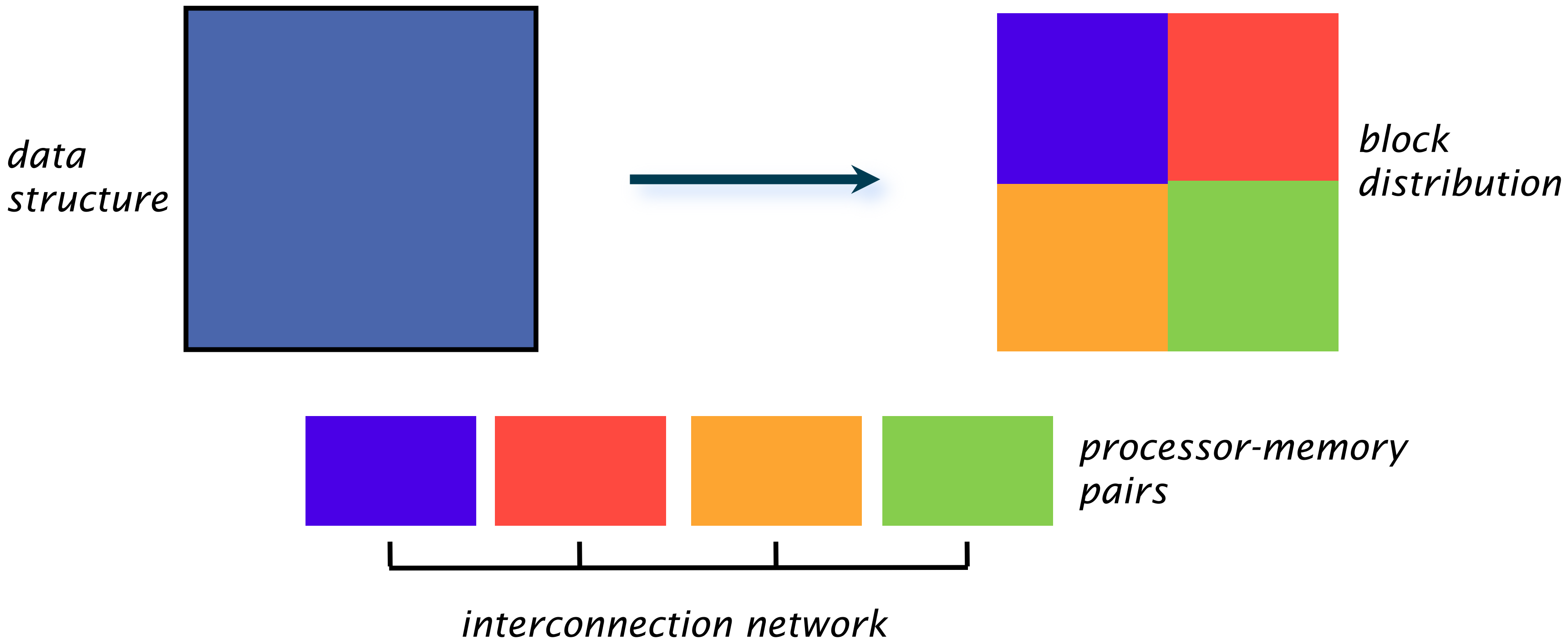
Divide and Conquer

- A task is recursively split in smaller tasks
- Fine grained tasks (i.e., leaves) are handed to workers
- Results are recursively aggregated towards the root
- Natural concurrency
  - different subproblems can be solved concurrently



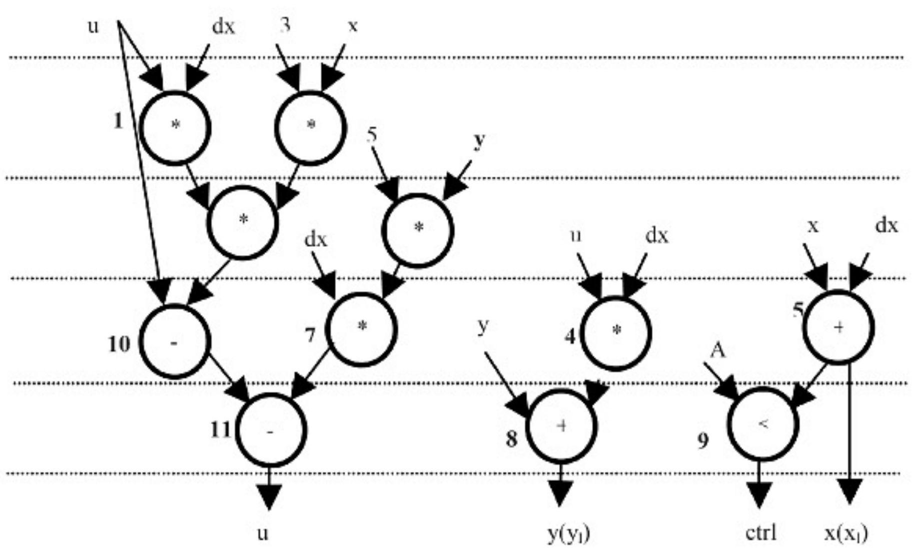
Data parallelism

- Data driven parallelization:
    - data is distributed to the memories of the processors.
- Computation follows the data



Task parallelism

- Distributes concurrent tasks to parallel computing nodes.
- Subclasses (by execution model):
  - pipelining
    - Different task per node
    - The intermediate results of the current task/node are used by the following node
  - data-flow (data-driven) execution
    - A task is executed as soon as all its input arguments are available
  - demand-driven execution
    - A task is only executed if its output is needed as input for another task



Fork-join model

- Threads are spawned into a “series-parallel” fashion
  - The OpenMP model
  - Can be seen as a “constrained” task-parallel model or a data-parallel model

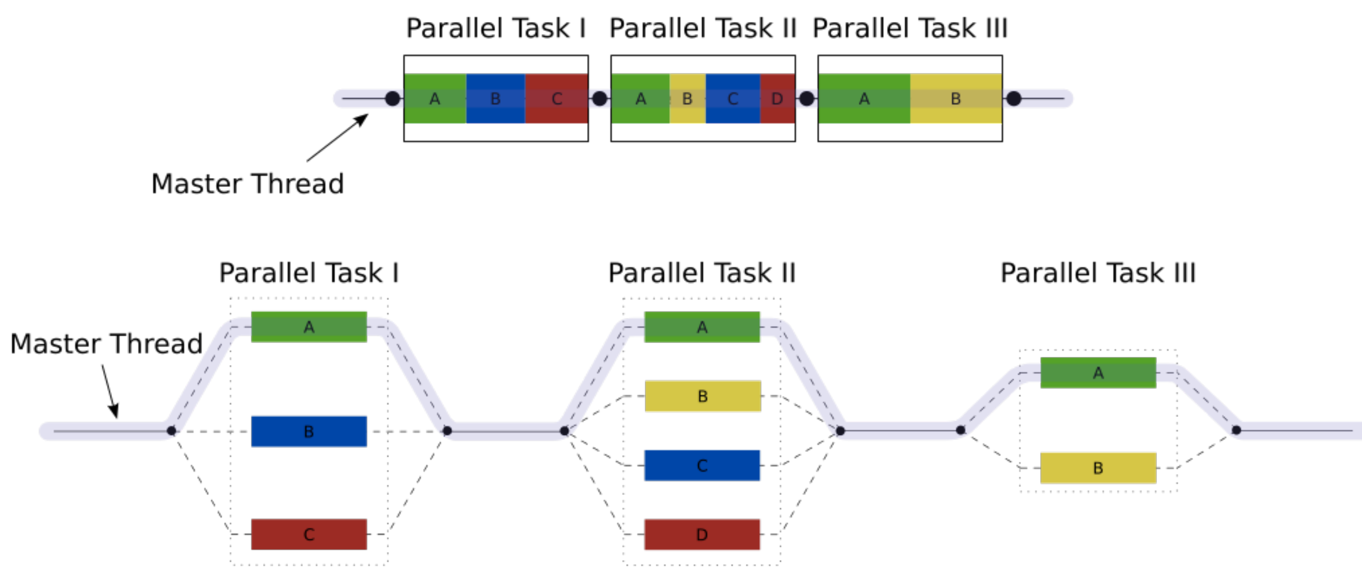
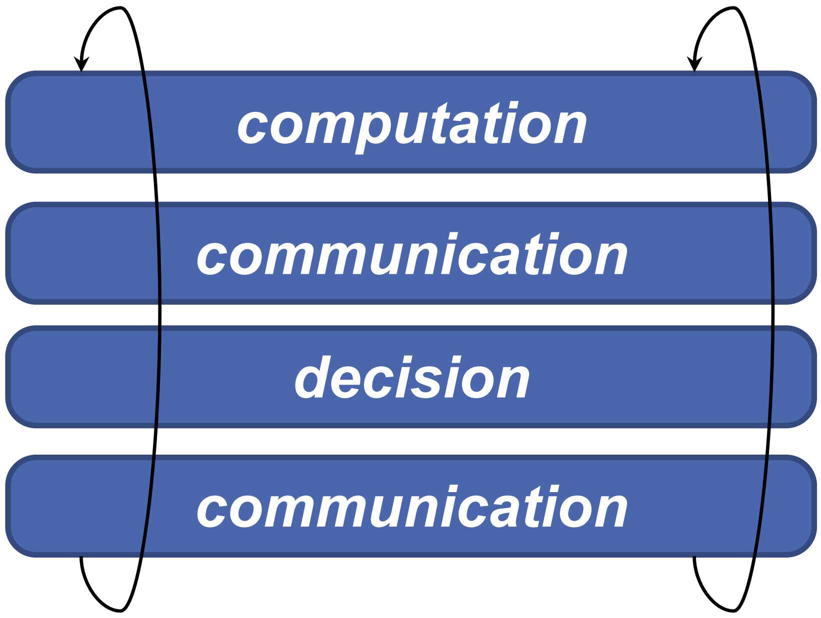


Image by Wikipedia user A1 - https://commons.wikimedia.org/w/index.php?curid=32004077

Conceptual: Bulk synchronous (BSP)

- Parallel computation consists of repeated cycles of
  - Compute phase, where all calculations are done locally
  - Communication phase (+ global synchronization)
  - Decision Phase (e.g. convergence)
  - Communication Phase (+ global synchronization)



Taking models into account

- Work = always the same
- Span = depending on the model

- Scheduling = depending on the model
  - Master/worker ?
  - Divide and conquer?
  - Task parallelism?

- What is the impact of scheduling on modeling?

Still to do …

- Estimate the number of operations
- Estimate the cost of each operation
- Validate our models