Concepts and Models of Parallel and Data-centric Programming

BSP III

Lecture, Summer 2020

Dr. Christian Terboven <terboven@itc.rwth-aachen.de>

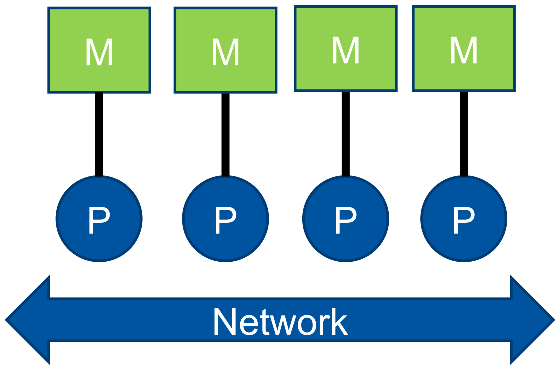
Outline

BSP Cost Model

BSP Model Components

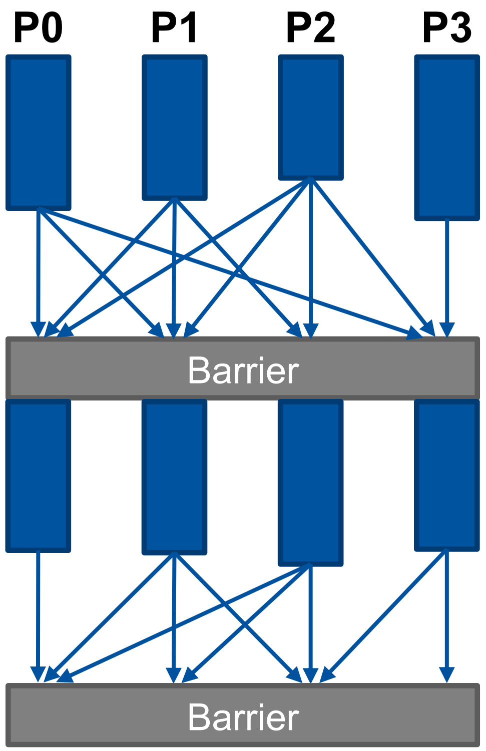
BSP Computer

(Distributed Memory Computer)

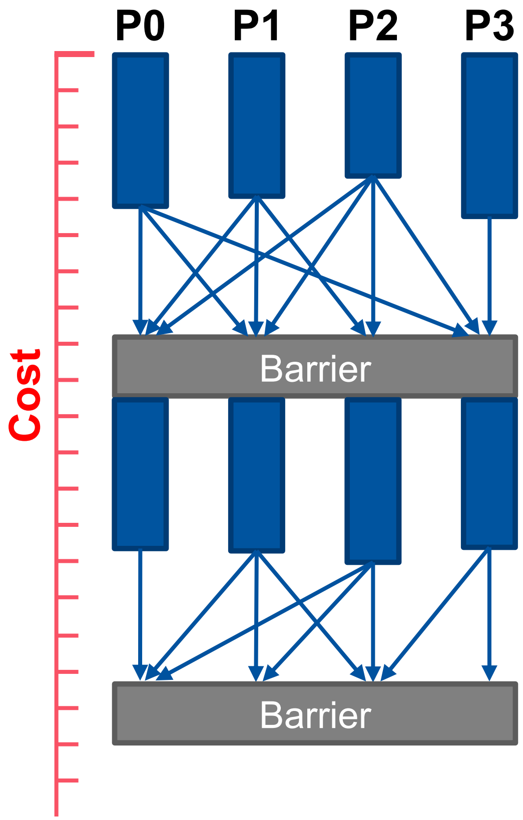


Programming Model

(Algorithmic Framework)

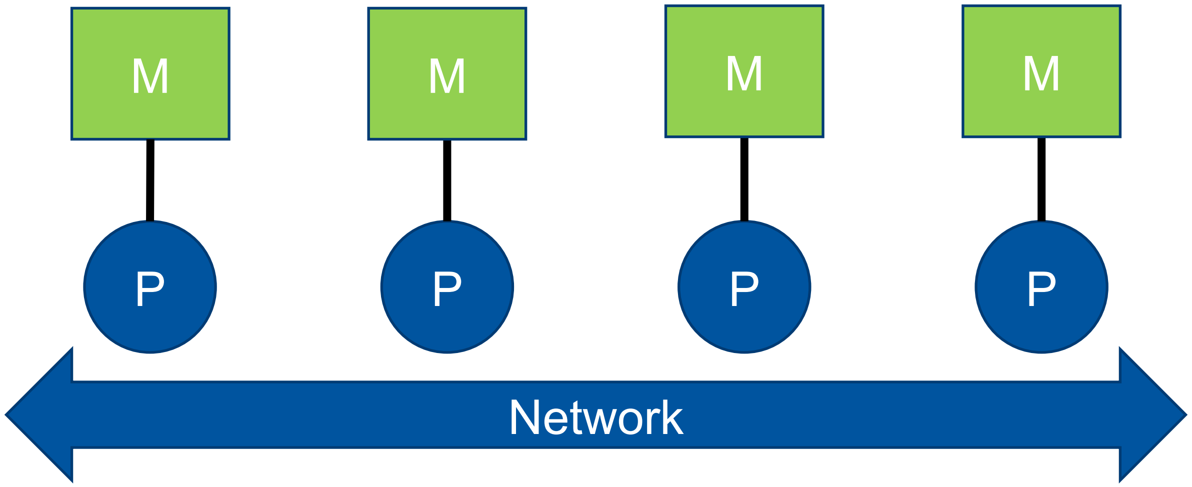


Cost Model



Cost Model

- BSP cost model: Predict runtime of algorithm running on BSP computer
- BSP computer is characterized as a 4-tuple (p, g, l, r)
  - p: Number of processors
  - g: Communication cost per data word (typical data word: double)
  - l: Latency (time a bulk synchronization takes)
  - r: Computation rate



Units

- BSP computer is characterized as a 4-tuple (p, g, l, r)
  - p: Number of processor (= number of *cores* in our terminology)
  - g: Communication cost per data word [s/word]
    - How long does it take to communicate a data word (on average)?
  - l: Latency [s]
    - How long does the barrier synchronization take?
  - r: Computation rate per processor [FLOP/s]
    - How many FLOPs can a single core compute per second?
- Convert FLOPs in time by division by r
- Convert time in FLOPs by multiplication with r

Example: CLAIX-2018 values

- MPI node
  - 2-socket (24 cores each) Intel Skylake
  - Interconnect: Intel Omni-Path
- Run microbenchmarks to get parameters
  - r stays constant (computation rate per core should stay the same)
  - g and l increase (due to increasing number of communication partners)

Parameter	1 Node	2 Nodes	4 Nodes	8 Nodes	16 Nodes
p	48	96	192	384	768
r [MFLOP/s]	565.3	563.7	563.1	564.6	564.0
g [ns/word]	19.86	26.65	28.23	32.05	39.45
l [ns]	43360	86120	198700	212000	961095

Cost of Computation Phase

- Count number of floating point operations (FLOPs) $w_i^{(s)}$ of processor s in superstep i
- Example: Assume following calculation of process s in superstep i (d, x, and y are doubles, j is an integer)

```
for (j := 0; j < n; j ++) do
    d[j] = a * x[j] + y[j]
```

2 FLOPs per iteration

  – Result: $w_i^{(s)} = 2n$ FLOPs for complete for-loop

- Time until all processors have finished their computation phase:

$$\frac{1}{r} \cdot \max_{0 \leq s < p} w_i^{(s)} \text{ [s]}$$

Computation rate [FLOP/s]

Cost of Communication Phase

- $r_i^{(s)}$: Number of data words received by process s in superstep i
- $t_i^{(s)}$: Number of data words transmitted by processor s in superstep i
- h-relation: $h_i = \max_{0 \leq s < p} \max\{r_i^{(s)}, t_i^{(s)}\}$ [Words]
  - Maximum number of words transmitted or received by *any* process in superstep i
  - Assumption: Processor can send *and* receive simultaneously
- Determine time until all other processors have finished communication phase by multiplying with g (communication cost per word):

$$g h_i \text{ [s]}$$

Cost of Superstep

- Putting computation phase and communication phase together
- Latency l for global synchronization overhead, initial send latency, …
- Time superstep i takes:

$$T_i = \underbrace{\frac{1}{r}\cdot \max_{0\le s<p} w_i^{(s)}}_{\text{Computation}} + \underbrace{g h_i}_{\text{Communication}} + \underbrace{l}_{\text{Latency}} \qquad [\mathrm{s}]$$

- **Note:** The cost model assumes a worst case: The most compute-intensive and the most communication-intensive phase are at the same processor.

the runtime provided the model cannot be always probably better than the runtime by actual program

BSP Costs: Parallel Inner Product

Input: x, y: vector of length n

$distr(x) = distr(y) = d$ with $d(i) = i \ mod \ p$ for $0 \leq i < n$ (cyclic)

Output: $\alpha = x^T y$

Algorithm for processor $s \in \{0, \dots, p-1\}$:

```
α_s := 0;
for (i := s; i < n; i += p) do
    α_s := α_s + x_i y_i;        // compute local product

for (t :=0; t < p ; t++) do
    put α_s in P(t);             // broadcast to all processors t
barrier();


α := 0;
for (t :=0; t < p; t++) do
    α := α + α_t;                // sum up local and received α values
```

Comp. Phase — $w_0^{(s)} = 2\lceil\frac{n}{p}\rceil$

Comm. Phase — $h_0 = p - 1$

Superstep 0

Comp. Phase — Superstep 1 — $w_1^{(s)} = p$

Cost Example: Inner Product

Superstep 0

- $w_0^{(s)} = 2\left\lceil\frac{n}{p}\right\rceil$
- $h_0 = p - 1$

$$\left.\right\} \begin{aligned} T_0 &= \frac{1}{r}\max_{0\le s<p} w_0^{(s)} + gh_0 + l \\ &= \frac{2}{r}\left\lceil\frac{n}{p}\right\rceil + g(p-1) + l \end{aligned}$$

Superstep 1

- $w_1^{(s)} = p$
- $h_1 = 0$ (no communication)

$$\left.\right\} \begin{aligned} T_1 &= \frac{1}{r}\max_{0\le s<p} w_1^{(s)} + gh_1 + l \\ &= \frac{p}{r} + l \end{aligned}$$

$$T = T_0 + T_1 = \frac{2}{r}\left\lceil\frac{n}{p}\right\rceil + \frac{p}{r} + g(p-1) + 2l$$

Example: Inner Product on CLAIX-2018 (1)

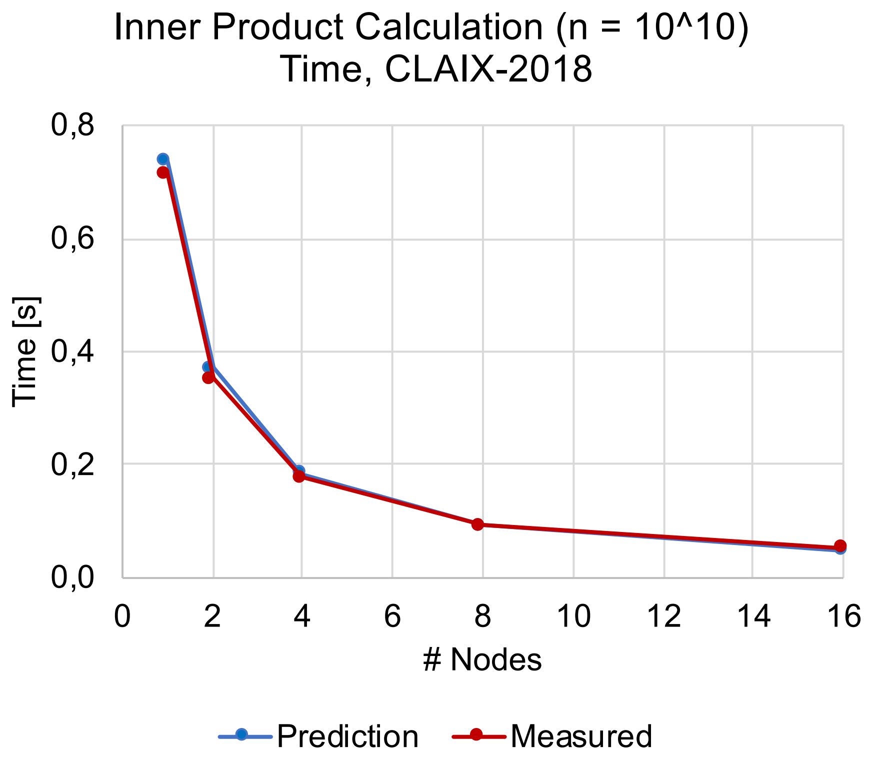
- Inner product of vectors with length n
  - $T = \frac{2}{r}\left\lceil\frac{n}{p}\right\rceil + \frac{p}{r} + g(p-1) + 2l$
- Assume: n = 10,000,000,000 (per node)

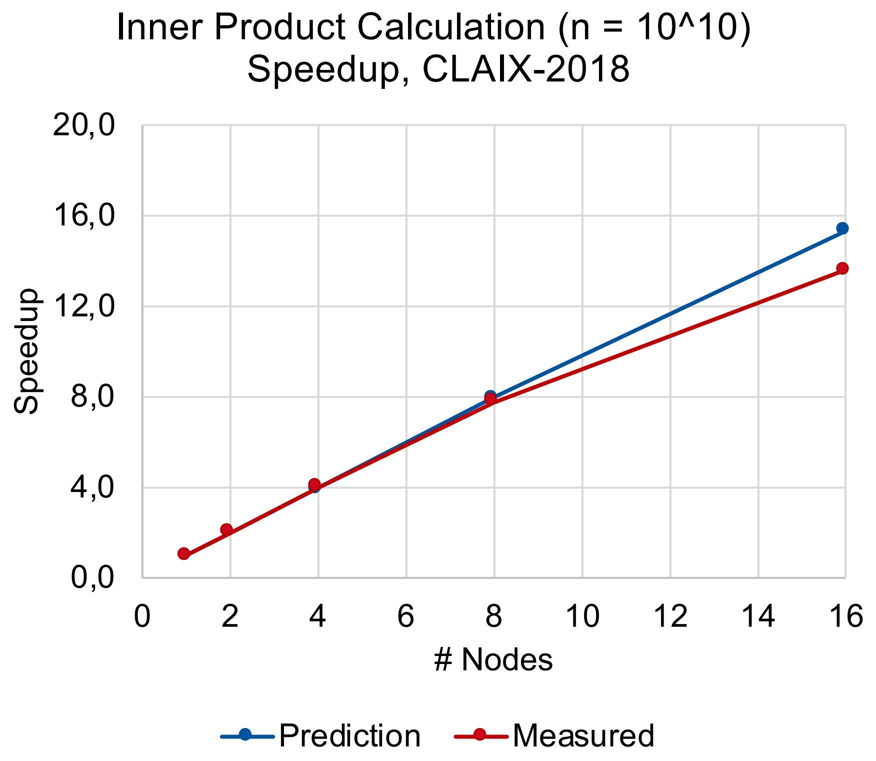
	1 Node	2 Nodes	4 Nodes	8 Nodes	16 Nodes
Predicted	0.7372	0.3697	0.1854	0.0927	0.0481
Measured	0.7142	0.3528	0.1783	0.0916	0.0525

- Predicted runtime differs from measured runtime 5 % – 10 %

Example: Inner Product on CLAIX-2018 (2)

Nodes	1	2	4	8	16
Predicted	0.7372	0.3697	0.1854	0.0927	0.0481
Measured	0.7142	0.3528	0.1783	0.0916	0.0525

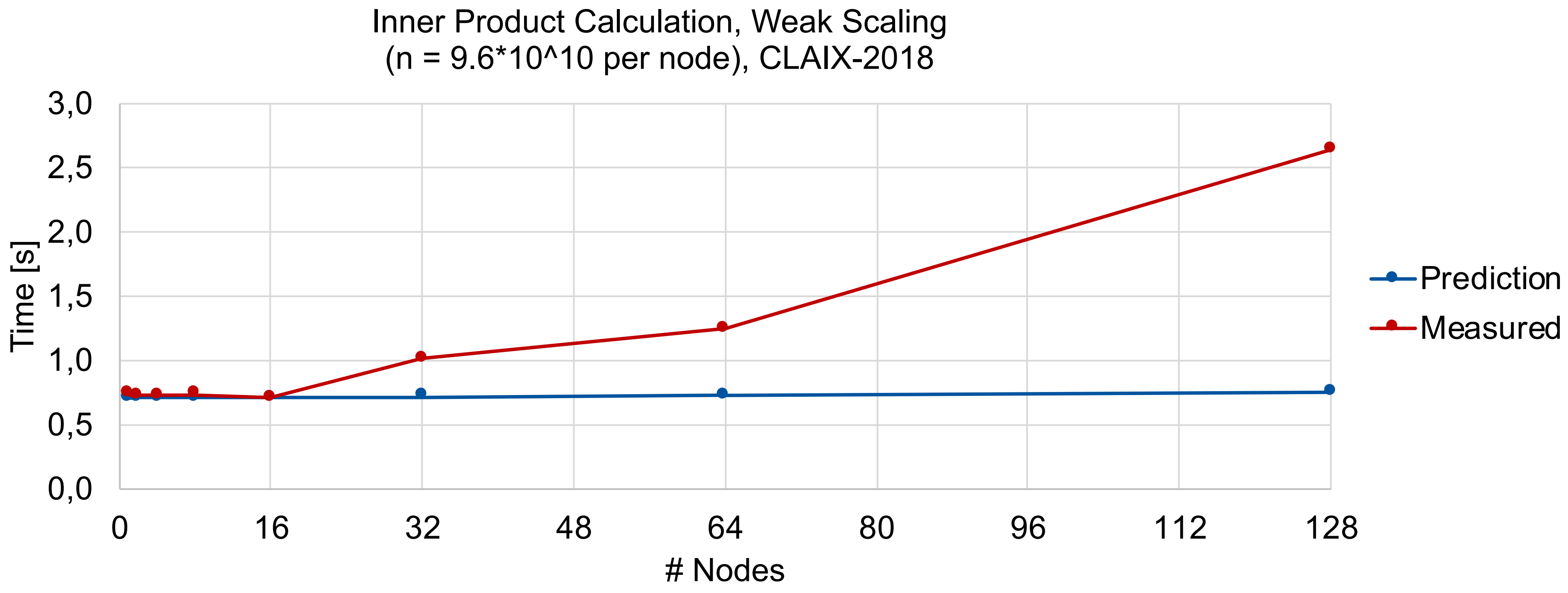




Example: Inner Product on CLAIX-2018 (3)

- Weak scaling experiment: 9.6*10^10 elements per node

Nodes	1	2	4	8	16	32	64	128
Prediction	0.7077	0.7078	0.7080	0.7080	0.7095	0.7173	0.7293	0.7536
Measured	0.7334	0.7237	0.7266	0.7394	0.7123	1.0150	1.2428	2.6421



Example: Inner Product on CLAIX-2018 (4)

- Observation: Cost model is precise for small node counts, but imprecise for large counts
  - Underestimates effect of communication and latencies
- Disclaimer: Locations of the selected nodes in the cluster (board, rack, …) were randomly assigned
  - Results for high node counts could be better if all compute nodes are allocated at the same rack (less communication costs)
  - Feature to select “nearby” nodes was not available when experiments were done

BSP Cost Model: What is Missing?

- Computation rate r (FLOP/s) significantly influences the result of the time prediction
- Problem: r heavily depends on executed code / algorithm and cannot be generally defined
  - What is the memory access pattern (cache vs. main memory accesses)?
  - Is the code / Are the loops vectorizable?
- Microbenchmark used here is a for loop with some FLOPs (addition, multiplication, subtraction)
  - Similar to innerproduct calculation → Prediction close to measured results

Microbenchmark for Computation Rate r

```cpp
double flop_rate() {
    unsigned int r_size = 1 << 23; // 2^23 elements
    std::vector<double> xs(r_size), ys(r_size), zs(r_size);

    // fill arrays

    auto alpha = // some value;
    auto beta = // some value;

    auto clock = bulk::util::timer();
    for (auto i = 0u; i < r_size; ++i) {
        zs[i] = zs[i] + alpha * xs[i] - beta * ys[i];
    }
    auto total_ms = clock.get();

    auto flops = r_size * 4; // 4*r_size FLOPS in loop
    auto flops_per_s = 1000.0 * flops / total_ms;
    return flops_per_s;
}
```

FLOP rate determined by for loop

BSP Cost Model: Summary

- Good way for a first cost estimation of a BSP algorithm
  - What limits my algorithm? Computation or communication?
  - But: Deviations are possible (see inner product benchmarks)
- **But:** Cost model does not consider any memory hierarchies (Caches, NUMA, …)
  - Estimation might differ from the actual result
- MultiBSP: Extension of BSP with a hierarchical execution model that consider different latencies of memories
  - Not discussed in this lecture
  - Further reading: Valiant, L.G.. A bridging model for multi-core computing. Journal of Computer and System Sciences 77, 1 (2011), 154 - 166 (https://doi.org/10.1016/j.jcss.2010.06.012)

What you have learnt

- Distributed Memory Computer / Cluster
  - Connecting shared memory machines via network together
- Communication in network can be two-sided or one-sided
- BSP model: Abstract model to structure parallel programming of large clusters
  - BSP computer
  - Programming model (algorithmic framework)
  - Cost model