Algorithms and scheduling techniques for clusters and grids

Yves Robert

École Normale Supérieure de Lyon
Yves.Robert@ens-lyon.fr
http://graal.ens-lyon.fr/~yrobert

joint work with
Olivier Beaumont, Anne Benoit, Larry Carter, Henri Casanova,
Jack Dongarra, Jeanne Ferrante, Arnaud Legrand, Loris Marchal, Frédéric Vivien

Murcia, May 26, 2008

Who cares about scheduling?

Heard it through the grapevine

Scheduling is “this thing that people in academia like to think about but that people who do real stuff sort of ignore”

Let’s prove this wrong?!

Evolution of parallel machines

From (good old) parallel architectures . . . . . . to heterogeneous clusters . . . . . . and to large-scale grid platforms?

Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines

On heterogeneous platforms, it gets worse

New platforms, new problems, **new solutions**

Target platforms: Large-scale heterogenous platforms (networks of workstations, clusters, collections of clusters, grids, ...)

New problems

- Heterogeneity of processors (CPU power, memory)
- Heterogeneity of communication links
- Irregularity of interconnection networks
- Non-dedicated platforms

Need to adapt algorithms and scheduling strategies: new objective functions, new models

Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms
2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications
3. Pipeline workflows
4. Models and real life
5. Conclusion

Independent chunks

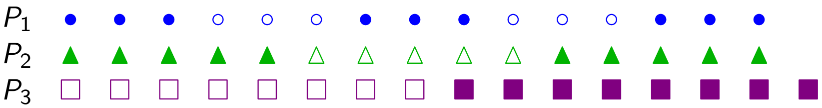
- B independent equal-size tasks • • • • • • • • • ••
- p processors $P_1, P_2, \ldots, P_p$
- w_i = time for P_i to process a task •

- **Intuition:** load of P_i proportional to its speed $1/w_i$
- Assign n_i tasks to P_i

Objective: minimize $\mathrm{T}_{\mathrm{exe}} = \max_{\sum_{i=1}^{p} n_i = B} (n_i \times w_i)$

Dynamic programming

With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$



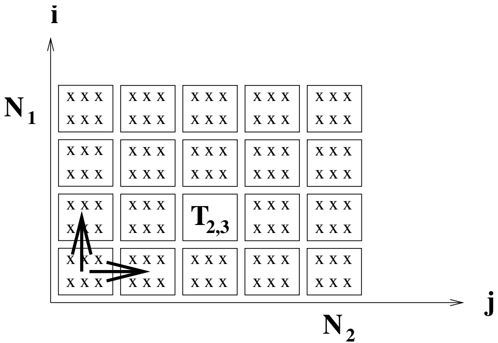
Task	n_1	n_2	n_3	T_{exe}	Selected proc.
0	0	0	0		1
1	1	0	0	3	2
2	1	1	0	5	1
3	2	1	0	6	3
4	2	1	1	8	1
5	3	1	1	9	2
6	3	2	1	10	1
7	4	2	1	12	1
8	5	2	1	15	2
9	5	3	1	15	3
10	5	3	2	16	

Static versus dynamic

- Greedy (demand-driven) would have done a perfect job
- Would even be better (possible variations in processor speeds)

Static assignment required **useless thinking** ☹

Coping with dependences

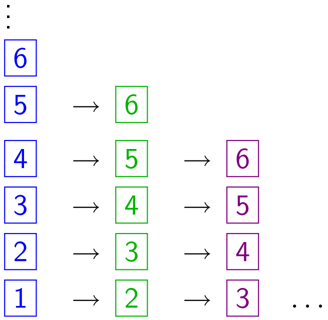


A simple finite difference problem

- Iteration space: 2D rectangle of size $N_1 \times N_2$
- Dependences between tiles $\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$

Allocation strategy (1/3)

Use column-wise allocation to enhance locality



Stepwise execution

Allocation strategy (2/3)

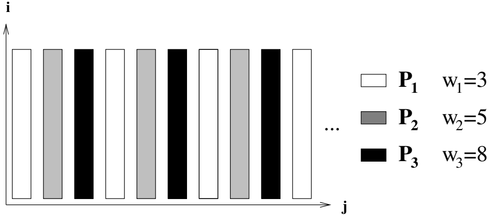
- With column-wise allocation,

$$T_{\text{opt}} \approx \frac{N_1 \times N_2}{\sum_{i=1}^{p} \frac{1}{w_i}}.$$

- Greedy (demand-driven) allocation $\Rightarrow$ **slowdown ?!**
- Execution progresses at the pace of the slowest processor ☹

Allocation strategy (3/3)

With 3 processors, $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$:

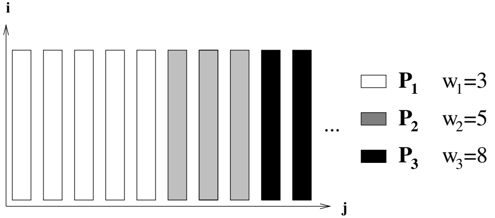


$$T_{exe} \approx \frac{8}{3} N_1 N_2 \approx 2.67\ N_1 N_2$$

$$T_{opt} \approx \frac{120}{79} N_1 N_2 \approx 1.52\ N_1 N_2$$

Periodic static allocation (1/2)

With 3 processors, $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$:



Assigning blocks of $B = 10$ columns, $\mathrm{T}_{\mathrm{exe}} \approx 1.6\ N_1 N_2$

Periodic static allocation (2/2)

- $L = \text{lcm}(w_1, w_2, \ldots, w_p)$
  Example: $L = \text{lcm}(3, 5, 8) = 120$
- P_1 receives first $n_1 = L/w_1$ columns, P_2 next $n_2 = L/w_2$ columns, and so on
- Period: block of $B = n_1 + n_2 + \ldots + n_p$ contiguous columns
  Example: $B = n_1 + n_2 + n_3 = 40 + 24 + 15 = 79$
- **Change schedule:**
  - Sort processors so that $n_1 w_1 \leq n_2 w_2 \leq \ldots \leq n_p w_p$
  - Process horizontally within blocks
- **Optimal** ☺

Lesson learnt?

With different-speed processors . . .
. . . we need to think (design static schedules)

. . . but implementation may remain dynamic ☺

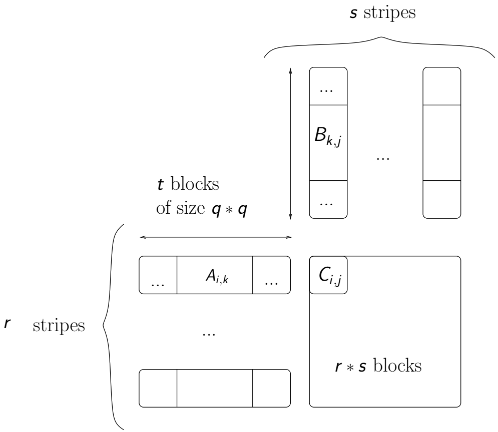
Example: demand-driven allocation of blocks of adequate size

. . . well, in some cases it gets truly complicated ☹

Why revisit matrix-product?

- A fundamental computational kernel (the mother of parallel algorithms)
- Archetype of a tightly-coupled application
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm

Application model

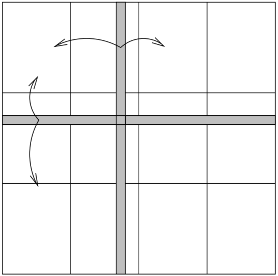


Use $q \times q$ blocks to harness efficiency of Level 3 BLAS

ScaLAPACK algorithm on (homogeneous) 2D grids (1/2)

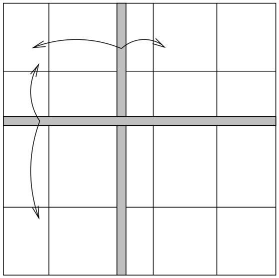
- $C = AB$ on a $p_1 \times p_2$ processor grid
- Granularity: one element = one square $q \times q$ block
- Each matrix is partitioned into $p_1 \times p_2$ rectangles
- Each processor is responsible for updating its rectangle
- Outer product version: at each step,
  - a column of blocks is communicated (broadcast) horizontally
  - a row of blocks is communicated (broadcast) vertically

ScaLAPACK algorithm on (homogeneous) 2D grids (2/2)



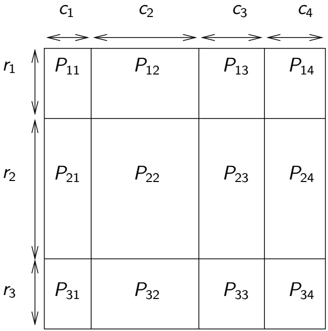
Matrix product on a 3×4 **homogeneous 2D-grid**

Algorithm on (heterogeneous) 2D grids (2/2)



Matrix product on a 3×4 **heterogeneous 2D-grid**

2D load balancing (1/2)



Objective: $\max_{r_i \times w_{ij} \times c_j \leq 1} \left\{ \left(\sum_{i=1}^{p_1} r_i \right) \times \left(\sum_{j=1}^{p_2} c_j \right) \right\}$

Maximize total number of elements processed within one time unit

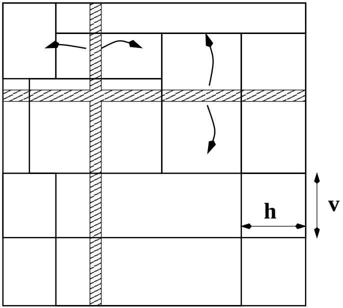
2D load balancing (2/2)

Given p processors, how to arrange them along a 2D grid of size $p_1 \times p_2 \leq p$ …

… so as to optimally load-balance the work of the processors

- Search among all possible arrangements of $p_1 \times p_2$ processors as a $p_1 \times p_2$ grid
- For each arrangement, solve optimization problem
- **NP-hard** ☹

Matrix product on heterogeneous clusters



Matrix product with 13 heterogeneous processors

Optimization

How to compute the *area* and *shape* of the p rectangles?

- **Load-balancing computations** assign *areas* proportional to speeds
- **Minimizing communication overhead** choose *shapes*:
  - total communication volume

$$\hat{C} = \sum_{i=1}^{p} (h_i + v_i)$$

  sum of the half perimeters of the p rectangles
  - for parallel communications:

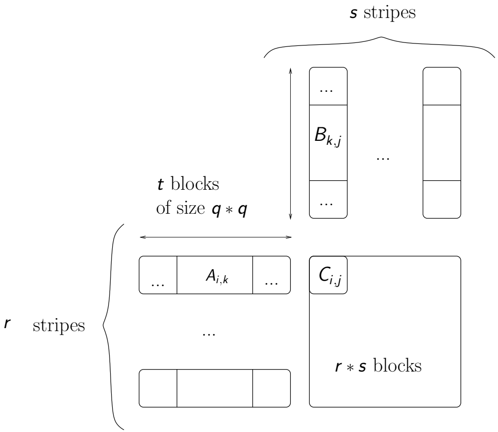
$$\hat{M} = \max_{i=1}^{p} (h_i + v_i)$$

- **Both problems NP-hard** ☹

Why revisit matrix-product?

- A fundamental computational kernel (the mother of parallel algorithms)
- Archetype of a tightly-coupled application
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
- **Target platforms = heterogeneous clusters**
- **Target usage = speed up MATLAB-client**

Same application model



Use $q \times q$ blocks to harness efficiency of Level 3 BLAS

Platform model

- *Star network* master M and p workers P_i
- $X.w_i$ time-units for P_i to execute a task of size X
- $X.c_i$ time-units for M to send/rcv msg of size X to/from P_i
- Master has no processing capability
- Enforce *one-port* model

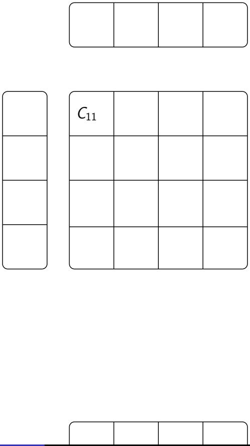
Memory limitation: only m_i buffers available for P_i
$\rightarrow$ at most m_i blocks simultaneously stored on worker

Strategy for allocating buffers

- Natural memory management
  - Assign one-third for each of $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$
  - **Example:** $m = 21 \Rightarrow 7$ buffers per matrix

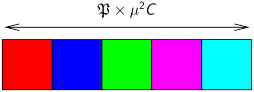
- Optimal memory management
  - Find largest μ s.t. $1 + \mu + \mu^2 \leq m$
  - Assign 1 buffer to $\mathcal{A}$, μ to $\mathcal{B}$ and μ^2 to $\mathcal{C}$
  - **Example:** $m = 21 \Rightarrow 1$ for $\mathcal{A}$, 4 to $\mathcal{B}$ and 16 to $\mathcal{C}$

Example with $m = 21$



Algorithm with identical workers

$c = 2,\ w = 4.5,\ \mu = 4,\ t = 100$, enroll $\mathfrak{P} = 5$ workers



$\mathfrak{P} \times \mu^2 C$ $\mathfrak{P} \times \mu(A, B)$

Performance

- Communication-to-computation ratio:

$$\frac{2}{t} + \frac{2}{\mu} \rightarrow \frac{2}{\sqrt{m}}$$

- Close to lower bound
- Enroll $\mathfrak{P} \leq p$ workers, where

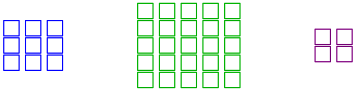
$$\mathfrak{P} = \left\lceil \frac{\mu w}{2c} \right\rceil$$

In the example, $\mathfrak{P} = \lceil 4.5 \rceil$

- Typically, $c = q^2 \tau_c$ and $w = q^3 \tau_a$
  $\rightarrow$ resource selection $\mathfrak{P} = \left\lceil \mu q \frac{\tau_a}{2\tau_c} \right\rceil$

Algorithms for heterogeneous platforms

- Different memory patterns for workers



- Complicated resource selection
- Complicated communication ordering
- Complicated schedule
- . . . but it works fine ☺ (see experiments in papers)

Lesson learnt?

Can provide efficient algorithms for tightly coupled applications but requires lots of efforts

. . . implementation cannot be demand-driven
unless ready to pay huge performance degradation

Example: resource selection plus static ordering mandatory for heterogeneous platforms

Iterative algorithms

Initial data (typically, a matrix)

Algorithm

1. Each processor performs a computation on its data chunk
2. Each processor exchanges the "border" of its data chunk of data with its neighbors
3. Go back to Step 1

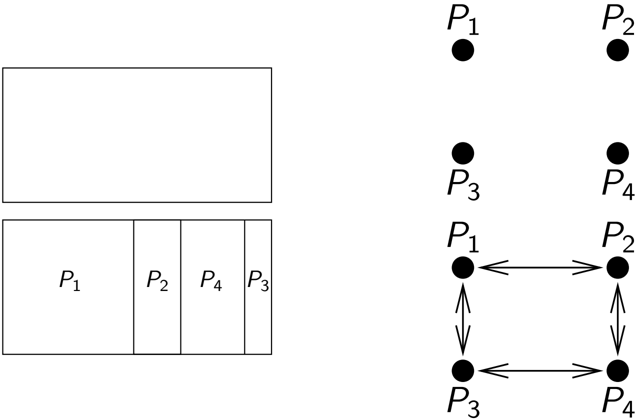
Questions

- Which processors should be used?
- What amount of data should they receive?
- How do we partition initial data set?

Impact of network models

Slicing data

- Data: a 2-D array



- Uni-dimensional partitioning into vertical slices
- Consequences:
  1. Borders and neighbors easily defined
  2. Constant volume of data exchanged between neighbors: D_c

Notations

- Processors: P_1, ..., P_p
- Processor P_i executes a unit task in time w_i
- Overall amount of work D_w;
  Share of P_i: $\alpha_i.D_w$ processed in time $\alpha_i.D_w.w_i$
  ($\alpha_i \geq 0$, $\sum_j \alpha_j = 1$)
- Cost of a unit-size communication from P_i to P_j: $c_{i,j}$
- Cost of a send from P_i to its successor in the ring: $D_c.c_{i,\text{succ}(i)}$

Communications: 1-port model

A processor can:

- send at most one message at any time
- receive at most one message at any time
- send and receive a message simultaneously

Objective

1. Select q processors out of p available resources
2. Arrange them along a ring
3. Distribute data

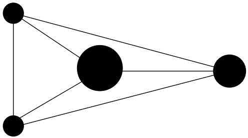
Minimize:

$$\max_{1 \leq i \leq p} \mathbb{I}\{i\}[\alpha_i . D_w . w_i + D_c . (c_{i,\text{pred}(i)} + c_{i,\text{succ}(i)})]$$

where $\mathbb{I}\{i\}[x] = 1$ if P_i participates in the computation, and 0 otherwise

Homogeneous fully-connected network

1. There exists a communication link between any processor pair
2. All links have same capacity
   ($\forall i, j \; c_{i,j} = c$)



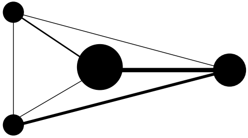
Results

- Either most powerful processor performs all the work, or all processors participate
- If all processors participate, all terminate work simultaneously
  $\alpha_i.D_w$ *rational values ???*
  $(\exists \tau, \quad \alpha_i.D_w.w_i = \tau$, so $1 = \sum_i \frac{\tau}{D_w.w_i})$
- Time of optimal solution:

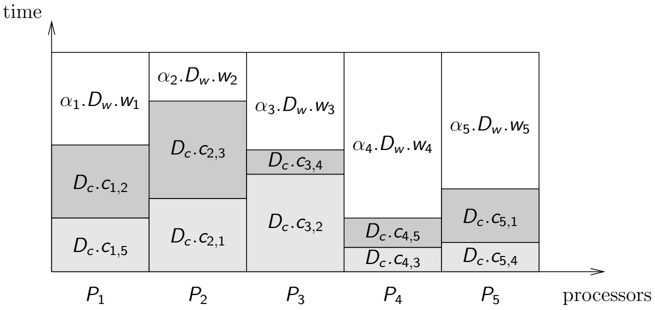
$$T_{\text{step}} = \min \left\{ D_w.w_{\min}, D_w.\frac{1}{\sum_i \frac{1}{w_i}} + 2.D_c.c \right\}$$

Heterogeneous fully-connected network

1. There exists a communication link between any processor pair
2. Links have different capacities



If all processors participate (1/3)



All processors end simultaneously

If all processors participate (2/3)

- All processors end simultaneously

$$T_{\text{step}} = \alpha_i . D_w . w_i + D_c . (c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)})$$

- $$\sum_{i=1}^{p} \alpha_i = 1 \;\Rightarrow\; \sum_{i=1}^{p} \frac{T_{\text{step}} - D_c . (c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)})}{D_w . w_i} = 1$$

$$\frac{T_{\text{step}}}{D_w . w_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}$$

where $w_{\text{cumul}} = \frac{1}{\sum_i \frac{1}{w_i}}$

If all processors participate (3/3)

$$\frac{T_{\text{step}}}{D_w . w_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}$$

$$T_{\text{step}} \text{ minimal} \Leftrightarrow \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i} \text{ is minimal}$$

Search an hamiltonian cycle of minimal weight in a graph where the edge from P_i to P_j has a weight of $d_{i,j} = \frac{c_{i,j}}{w_i} + \frac{c_{j,i}}{w_j}$

NP-complete problem

If all processors participate: linear program

$$\text{MINIMIZE } \sum_{i=1}^{p}\sum_{j=1}^{p} d_{i,j}.x_{i,j},$$

SATISFYING THE (IN)EQUATIONS

$$\begin{cases}
(1)\ \sum_{j=1}^{p} x_{i,j} = 1 & 1 \leq i \leq p \\
(2)\ \sum_{i=1}^{p} x_{i,j} = 1 & 1 \leq j \leq p \\
(3)\ x_{i,j} \in \{0,1\} & 1 \leq i,j \leq p \\
(4)\ u_i - u_j + p.x_{i,j} \leq p-1 & 2 \leq i,j \leq p, i \neq j \\
(5)\ u_i \text{ integer}, u_i \geq 0 & 2 \leq i \leq p
\end{cases}$$

$x_{i,j} = 1$ if, and only if, the edge from P_i to P_j is used

General case : linear program

Best ring made of q processors

MINIMIZE T SATISFYING THE (IN)EQUATIONS

$$
\left\{
\begin{array}{lll}
(1) & x_{i,j} \in \{0,1\} & 1 \leq i,j \leq p \\
(2) & \sum_{i=1}^{p} x_{i,j} \leq 1 & 1 \leq j \leq p \\
(3) & \sum_{i=1}^{p} \sum_{j=1}^{p} x_{i,j} = q & \\
(4) & \sum_{i=1}^{p} x_{i,j} = \sum_{i=1}^{p} x_{j,i} & 1 \leq j \leq p \\
& & \\
(5) & \sum_{i=1}^{p} \alpha_i = 1 & \\
(6) & \alpha_i \leq \sum_{j=1}^{p} x_{i,j} & 1 \leq i \leq p \\
(7) & \alpha_i.w_i + \frac{D_c}{D_w} \sum_{j=1}^{p} (x_{i,j} c_{i,j} + x_{j,i} c_{j,i}) \leq T & 1 \leq i \leq p \\
& & \\
(8) & \sum_{i=1}^{p} y_i = 1 & \\
(9) & -p.y_i - p.y_j + u_i - u_j + q.x_{i,j} \leq q - 1 & 1 \leq i,j \leq p, i \neq j \\
(10) & y_i \in \{0,1\} & 1 \leq i \leq p \\
(11) & u_i \text{ integer}, u_i \geq 0 & 1 \leq i \leq p
\end{array}
\right.
$$

Linear programming

- Problems with rational variables: can be solved in polynomial time (in the size of the problem)
- Problems with integer variables: solved in exponential time in the worst case
- No relaxation in rational numbers seems possible here...

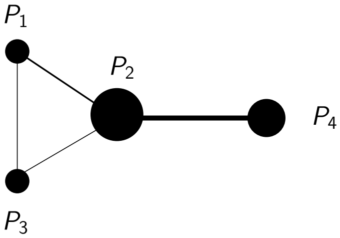
And, in practice ?

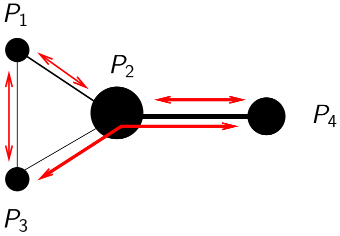
If all processors participate. Use a heuristic to solve the traveling salesman problem (as Lin-Kernighan)
No guarantee, but excellent results in practice.

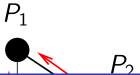
General case.

1. Exhaustive search: feasible up to a dozen of processors
2. Greedy heuristic:
   - initially take best pair of processors
   - for a given ring, try to insert any unused processor in between any pair of neighbor processors in the ring

Heterogeneous network (general case)



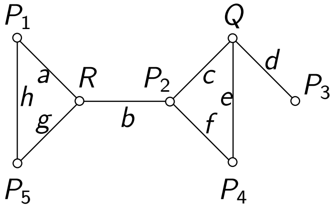




New notations

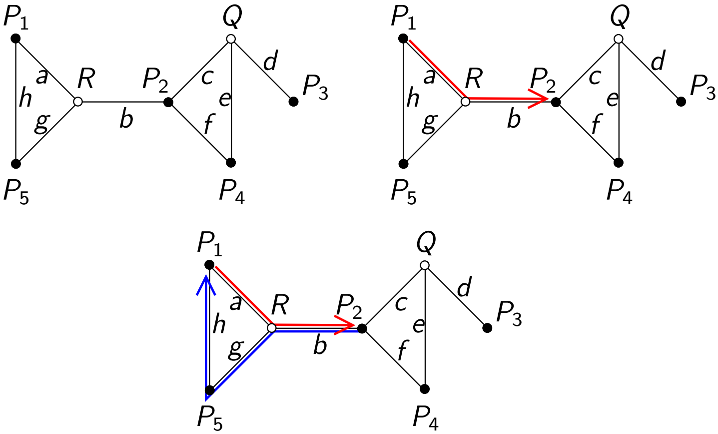
- Set of communications links: e_1, ..., e_n
- Bandwidth of link e_m: b_{e_m}
- There is a path $\mathcal{S}_i$ from P_i to $P_{\text{succ}(i)}$ in the network
  - $\mathcal{S}_i$ uses a fraction $s_{i,m}$ of the bandwidth b_{e_m} of link e_m
  - P_i needs a time $D_c . \dfrac{1}{\min_{e_m \in \mathcal{S}_i} s_{i,m}}$ to send a message of size D_c to its successor
  - Constraints on the bandwidth of e_m: $\displaystyle\sum_{1 \leq i \leq p} s_{i,m} \leq b_{e_m}$
- Symmetrically, there is a path $\mathcal{P}_i$ from P_i to $P_{\text{pred}(i)}$ in the network, which uses a fraction $p_{i,m}$ of the bandwidth b_{e_m} of link e_m

Toy example: choosing the ring



- 7 processors and 8 bidirectional communications links
- We choose a ring of 5 processors:
  $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5$ (we use neither Q, nor R)

Toy example: choosing routing paths



From P_1 to P_2, use links a and b: $\mathcal{S}_1 = \{a, b\}$.

From P_2 to P_1, use links b, g and h: $\mathcal{P}_2 = \{b, g, h\}$.

From P_1: to P_2, $\mathcal{S}_1 = \{a, b\}$ and to P_5, $\mathcal{P}_1 = \{h\}$
From P_2: to P_3, $\mathcal{S}_2 = \{c, d\}$ and to P_1, $\mathcal{P}_2 = \{b, g, h\}$

Toy example: bandwidth sharing

From P_1 to P_2 we use links a and b: $c_{1,2} = \frac{1}{\min(s_{1,a}, s_{1,b})}$.
From P_1 to P_5 we use link h: $c_{1,5} = \frac{1}{p_{1,h}}$.

Set of all sharing constraints:

Lien a: $s_{1,a} \leq b_a$
Lien b: $s_{1,b} + s_{4,b} + p_{2,b} + p_{5,b} \leq b_b$
Lien c: $s_{2,c} \leq b_c$
Lien d: $s_{2,d} + s_{3,d} + p_{3,d} + p_{4,d} \leq b_d$
Lien e: $s_{3,e} + p_{3,e} + p_{4,e} \leq b_e$
Lien f: $s_{4,f} + p_{3,f} + p_{5,f} \leq b_f$
Lien g: $s_{4,g} + p_{2,g} + p_{5,g} \leq b_g$
Lien h: $s_{5,h} + p_{1,h} + p_{2,h} \leq b_h$

Toy example: final quadratic system

MINIMIZE $\max_{1 \leq i \leq 5} \left(\alpha_i.D_w.w_i + D_c.(c_{i,i-1} + c_{i,i+1})\right)$ UNDER THE CONSTRAINTS

$$
\left\{
\begin{array}{lll}
\sum_{i=1}^{5} \alpha_i = 1 & & \\
s_{1,a} \leq b_a & s_{1,b} + s_{4,b} + p_{2,b} + p_{5,b} \leq b_b & s_{2,c} \leq b_c \\
s_{2,d} + s_{3,d} + p_{3,d} + p_{4,d} \leq b_d & s_{3,e} + p_{3,e} + p_{4,e} \leq b_e & s_{4,f} + p_{3,f} + p_{5,f} \leq b_f \\
s_{4,g} + p_{2,g} + p_{5,g} \leq b_g & s_{5,h} + p_{1,h} + p_{2,h} \leq b_h & \\
s_{1,a}.c_{1,2} \geq 1 & s_{1,b}.c_{1,2} \geq 1 & p_{1,h}.c_{1,5} \geq 1 \\
s_{2,c}.c_{2,3} \geq 1 & s_{2,d}.c_{2,3} \geq 1 & p_{2,b}.c_{2,1} \geq 1 \\
p_{2,g}.c_{2,1} \geq 1 & p_{2,h}.c_{2,1} \geq 1 & s_{3,d}.c_{3,4} \geq 1 \\
s_{3,e}.c_{3,4} \geq 1 & p_{3,d}.c_{3,2} \geq 1 & p_{3,e}.c_{3,2} \geq 1 \\
p_{3,f}.c_{3,2} \geq 1 & s_{4,f}.c_{4,5} \geq 1 & s_{4,b}.c_{4,5} \geq 1 \\
s_{4,g}.c_{4,5} \geq 1 & p_{4,e}.c_{4,3} \geq 1 & p_{4,d}.c_{4,3} \geq 1 \\
s_{5,h}.c_{5,1} \geq 1 & p_{5,g}.c_{5,4} \geq 1 & p_{5,b}.c_{5,4} \geq 1 \\
p_{5,f}.c_{5,4} \geq 1 & &
\end{array}
\right.
$$

Toy example: the moral

Problem sums up to a quadratic system if

1. processors are already selected
2. processors are already ordered into a ring
3. communication paths are already known

In other words: a quadratic system if the ring is known.
If the ring is known:

- Complete graph: closed-form expression
- General graph: quadratic system

Is the more complex network model **with link contention** worth the trouble?

And, in practice ?

Adapt greedy heuristic:

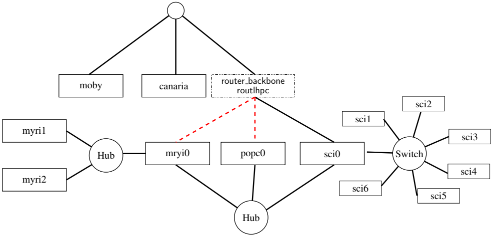
1. Initially: best processor pair
2. For each processor P_k (not already included in the ring)
   - For each pair (P_i, P_j) of neighbors in the ring
     1. Build graph of unused bandwidths (without considering the paths between P_i and P_j)
     2. Compute shortest paths (in terms of bandwidth) between P_k and P_i and P_j
     3. Evaluate solution
3. Keep best solution found at step 2 and start again

+ refinements (*max-min fairness*, quadratic solving)

Is this meaningful ?

- No guarantee, neither theoretical, nor practical
- Simple solution:
  1. build complete graph whose edges are labeled with bandwidths of best communication paths
  2. apply the heuristic for complete graphs
  3. allocate bandwidths

An example of an actual platform (Lyon)



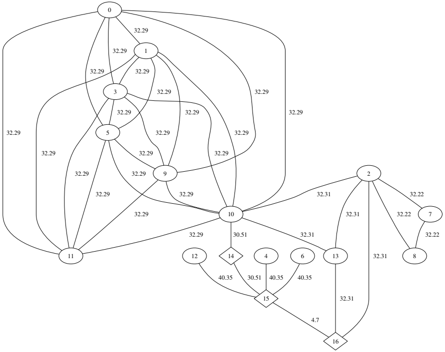
Topology

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
0.0206	0.0206	0.0206	0.0206	0.0291	0.0206	0.0087	0.0206	0.0206

P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}
0.0206	0.0206	0.0206	0.0291	0.0451	0	0	0

Processors processing times (in seconds par megaflop)

Describing the Lyon platform



Abstracting the Lyon platform.

Results

First heuristic building the ring without taking link sharing into account

Second heuristic taking link sharing into account (and with quadratic programming)

Ratio D_c/D_w	H1	H2	Gain
0.64	0.008738 (1)	0.008738 (1)	0%
0.064	0.018837 (13)	0.006639 (14)	64.75%
0.0064	0.003819 (13)	0.001975 (14)	48.28%

Ratio D_c/D_w	H1	H2	Gain
0.64	0.005825 (1)	0.005825 (1)	0 %
0.064	0.027919 (8)	0.004865 (6)	82.57%
0.0064	0.007218 (13)	0.001608 (8)	77.72%

Table: T_{step}/D_w for each heuristic on the Lyon and Strasbourg platforms (numbers in parentheses show size of solution rings)

And with non dedicated platforms?

Available processing power of each processor changes over time

Available bandwidth of each communication link changes over time

$\Rightarrow$ Need to reconsider current allocation

$\Rightarrow$ Introduce (dynamic) redistribution algorithms

A possible approach

- If actual performance "too much" different from expected characteristics when building solution

  Actual criterion defining "too much" ?

  - If actual performance "very" different
    - compute a new ring
    - redistribute data from old ring to new one

      Actual criterion defining "very" ?

      Cost of the redistribution ?
  - If the actual performance is "a little" different
    - compute new load-balancing in existing ring
    - redistribute data in existing ring

      How to efficiently do the redistribution ?

Load-balancing

Principle: ring is modified only if this is profitable

- T_{step}: length of an iteration *before* load-balancing
- T'_{step}: length of an iteration *after* load-balancing
- $T_{\text{redistribution}}$: redistribution cost
- n_{iter}: number of remaining iterations

Condition: $$T_{\text{redistribution}} + n_{\text{iter}} \times T'_{\text{step}} \leq n_{\text{iter}} \times T_{\text{step}}$$

Redistribution algorithms for homo/hetero uni/bi-dir rings

(Well, let's do this another time ...)

Lesson learnt?

Realistic networks models: mandatory but less tractable

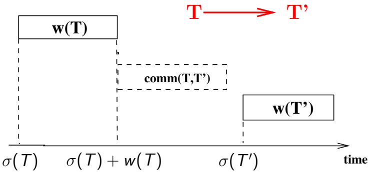
. . . Need find good trade-offs.

Would be even more complicated with hierarchical architectures ☹

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - $\text{alloc}(T)$ = processor assigned to it

Traditional scheduling – Constraints



- Data dependences If $(T, T') \in E$ then
  - if $\text{alloc}(T) = \text{alloc}(T')$ then $\sigma(T) + w(T) \leq \sigma(T')$
  - if $\text{alloc}(T) \neq \text{alloc}(T')$ then $\sigma(T) + w(T) + c(T, T') \leq \sigma(T')$
- Resource constraints

$$\text{alloc}(T) = \text{alloc}(T') \Rightarrow$$
$$(\sigma(T) + w(T) \leq \sigma(T')) \text{ or } (\sigma(T') + w(T') \leq \sigma(T))$$

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + w(T))$$

- Other classical objectives:
  - Sum of completion times
  - With release dates: max flow (response time), or sum flow
  - Fairness oriented: max stretch, or sum stretch

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unlimited number of simultaneous sends/receives per processor
  - No contention $\rightarrow$ unbounded bandwidth on any link
  - Fully connected interconnection graph (clique)
- In fact, model assumes infinite network capacity

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications
    ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

List scheduling – Without communications

Initialization:

- Compute priority level of all tasks
- Priority queue = list of free tasks (tasks without predecessors) sorted by priority

While there remain tasks to execute:

- Add new free tasks, if any, to the queue.
- If there are q available processors and r tasks in the queue, remove first $\min(q, r)$ tasks from the queue and execute them

Priority level

- Use critical path: longest path from the task to an exit node
- Computed recursively by a bottom-up traversal of the graph

List scheduling – With communications (1/2)

- Priority level
  - Use *pessimistic* critical path: include all edge costs in the weight
  - Computed recursively by a bottom-up traversal of the graph

- MCP *Modified Critical Path*
  - Assign free task with highest priority to *best* processor
  - Best processor = finishes execution first, given already taken scheduling decisions
  - Free tasks may not be ready for execution (communication delays)
  - May explore inserting the task in empty slots of schedule
  - Complexity $O(|V| \log |V| + (|E| + |V|)p)$

List scheduling – With communications (2/2)

- EFT *Earliest Finish Time*
  - Dynamically recompute priorities of free tasks
  - Select free task that finishes execution first (on best processor), given already taken scheduling decisions
  - Higher complexity $O(|V|^3 p)$
  - May miss "urgent" tasks on critical path
- Other approaches
  - Two-step: clustering + load balancing
    - DSC Dominant Sequence Clustering $O((|V| + |E|) \log |V|)$
    - LLB List-based Load Balancing $O(C \log C + |V|)$ (C number of clusters generated by DSC)
  - Low-cost: FCP Fast Critical Path
    - Maintain constant-size sorted list of free tasks:
    - Best processor = first idle or the one sending last message
    - Low complexity $O(|V| \log p + |E|)$

Extending the model to heterogeneous clusters

- Task graph with n tasks $T_1, \ldots, T_n$.
- Platform with p heterogeneous processors $P_1, \ldots, P_p$.
- Computation costs:
  - w_{iq} = execution time of T_i on P_q
  - $\overline{w_i} = \frac{\sum_{q=1}^{p} w_{iq}}{p}$ average execution time of T_i
  - particular case: consistent tasks $w_{iq} = w_i \times \gamma_q$
- Communication costs:
  - data(i,j): data volume for edge $e_{ij} : T_i \rightarrow T_j$
  - v_{qr}: communication time for unit-size message from P_q to P_r (zero if $q = r$)
  - com(i,j,q,r) = data$(i,j) \times v_{qr}$ communication time from T_i executed on P_q to P_j executed on P_r
  - $\overline{\text{com}_{ij}}$ = data$(i,j) \times \frac{\sum_{1 \leq q,r \leq p, q \neq r} v_{qr}}{p(p-1)}$ average communication cost for edge $e_{ij} : T_i \rightarrow T_j$

Rewriting constraints

Dependences For $e_{ij} : T_i \rightarrow T_j$, $q = \text{alloc}(T_i)$ and $r = \text{alloc}(T_j)$:

$$\sigma(T_i) + w_{iq} + \text{com}(i,j,q,r) \leq \sigma(T_j)$$

Resources If $q = \text{alloc}(T_i) = \text{alloc}(T_j)$, then

$$(\sigma(T_i) + w_{iq} \leq \sigma(T_j)) \text{ or } (\sigma(T_j) + w_{jq} \leq \sigma(T_i))$$

Makespan

$$\max_{1 \leq i \leq n} \left(\sigma(T_i) + w_{i,\text{alloc}(T_i)}\right)$$

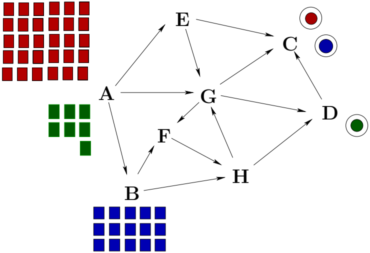
HEFT: Heterogeneous Earliest Finish Time

- Priority level:
  - $\text{rank}(T_i) = \overline{w_i} + \max_{T_j \in \text{Succ}(T_i)} (\overline{\text{com}_{ij}} + \text{rank}(T_j))$,
    where $\text{Succ}(T)$ is the set of successors of T
  - Recursive computation by bottom-up traversal of the graph
- Allocation
  - For current task T_i, determine best processor P_q: minimize $\sigma(T_i) + w_{iq}$
  - Enforce constraints related to communication costs
  - Insertion scheduling: look for $t = \sigma(T_i)$ s.t. P_q is available during interval $[t, t + w_{iq}[$
- Complexity: same as MCP without/with insertion

What's wrong?

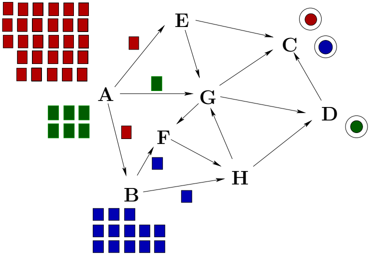
- ☺ Nothing (still may need to map a DAG onto a platform!)
- ☹ Absurd communication model:
  complicated: many parameters to instantiate
  while not realistic (clique + no contention)
- ☹ Wrong metric: need to relax makespan minimization objective

Problem



- Routing sets of messages from sources to destinations
- Paths not fixed a priori
- Packets of same message may follow different paths

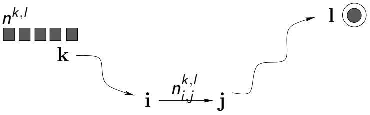
Hypotheses



- A packet crosses an edge within one time-step
- At any time-step, at most one packet crosses an edge

Scheduling: for each time-step, decide which packet crosses any given edge

Notation



- $n^{k,l}$: total number of packets to be routed from k to l
- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i, j)

Lower bound

Congestion $C_{i,j}$ of edge (i,j)
$=$ total number of packets that cross (i,j)

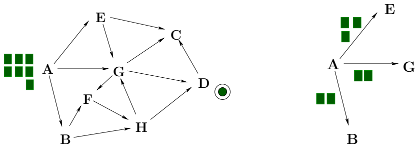
$$C_{i,j} = \sum_{(k,l) \mid n^{k,l} > 0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

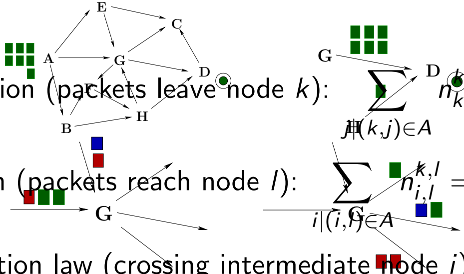
$C_{\max}$ lower bound on schedule makespan

$$C^* \geq C_{\max}$$

$\Rightarrow$ "Fluidified" solution in $C_{\max}$?

Equations (1/2)





- Initialization (packets leave node k): $\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$
- Reception (packets reach node l): $\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$
- Conservation law (crossing intermediate node i):

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (2/2)

- Congestion

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l}$$

- Objective function

$$C_{\max} \geq C_{i,j}, \qquad \forall i, j$$

$$\text{Minimize } C_{\max}$$

Linear program in rational numbers: polynomial-time solution. In practice use GLPK, Maple, Mupad ...

Routing algorithm

- Compute optimal solution $C_{\max}$, $n_{i,j}^{k,l}$ of previous linear program
- Periodic schedule:
  - Define $\Omega = \sqrt{C_{\max}}$
  - Use $\left\lceil \frac{C_{\max}}{\Omega} \right\rceil$ periods of length Ω
  - During each period, edge (i,j) forwards (at most)

$$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

    packets that go from k to l
- Clean-up: sequentially process residual packets inside network

Performance

- Schedule is feasible
- Schedule is asymptotically optimal:

$$C_{\max} \leq C^* \leq C_{\max} + O(\sqrt{C_{\max}})$$

Why does it work?

- Relaxation of objective function
- Rational number of packets in LP formulation
- Periods long enough so that rounding down to integer numbers has negligible impact
- Periods numerous enough so that loss in first and last periods has negligible impact
- Periodic schedule, described in compact form

Master-worker tasking: framework

Heterogeneous resources

- Processors of different speeds
- Communication links with various bandwidths

Large number of independent tasks to process

- Tasks are atomic
- Tasks have same size

Single data repository

- One master initially holds data for all tasks
- Several workers arranged along a star, a tree or a general graph

Application examples

- Monte Carlo methods
- SETI@home
- Factoring large numbers
- Searching for Mersenne primes
- Particle detection at CERN (LHC@home)
- ... and many others: see BOINC at `http://boinc.berkeley.edu`

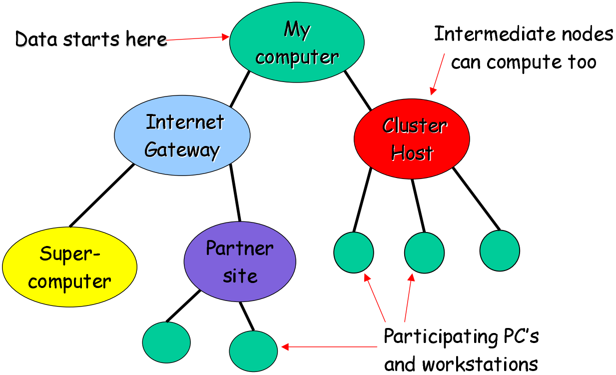
Makespan vs. steady state

Two-different problems

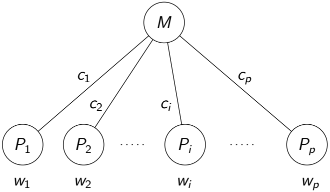
Makespan Maximize total number of tasks processed within a time-bound

Steady state Determine **periodic task allocation** which maximizes total throughput

Example


Data starts here
My computer
Intermediate nodes can compute too
Internet Gateway
Cluster Host
Super-computer
Partner site
Participating PC's and workstations

Rule of the gameEquations



- Master sends tasks to workers sequentially, and without preemption
- Full computation/communication overlap for each worker
- Worker P_i receives a task in c_i time-units
- Worker P_i processes a task in w_i time-units
- Worker P_i executes α_i tasks per time-unit
- Computations: $\alpha_i w_i \leq 1$

Solution

- Faster-communicating workers first: $c_1 \leq c_2 \leq \ldots$
- Make full use of first q workers, where q largest index s.t.

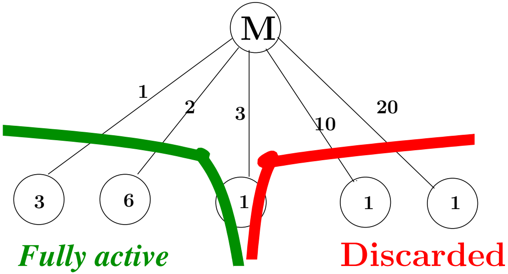
$$\sum_{i=1}^{q} \frac{c_i}{w_i} \leq 1$$

- Make partial use of next worker P_{q+1}
- **Discard** other workers

Bandwidth-centric strategy

- Delegate work to the fastest communicating workers
- It doesn't matter if these workers are computing slowly
- Slow workers will not contribute much to overall throughput

Example

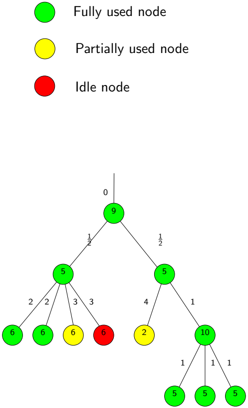


Tasks	Communication	Computation
6 tasks to P_1	$6c_1 = 6$	$6w_1 = 18$
3 tasks to P_2	$3c_2 = 6$	$3w_2 = 18$
2 tasks to P_3	$2c_3 = 6$	$2w_3 =$ **2**

11 tasks every 18 time-units ($\rho = 11/18 \approx 0.6$)

☺ Compare to purely greedy (demand-driven) strategy!

Extension to trees

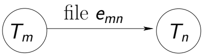


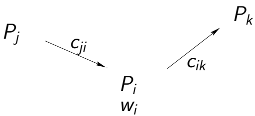
Resource selection based on **local** information (children)

Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)? Yes, I mean, almost!

LP formulation still works well . . .





Conservation law

$$\forall m, n \quad \sum_j \text{sent}(P_j \to P_i, e_{mn}) + \text{executed}(P_i, T_m) = \text{executed}(P_i, T_n) + \sum_k \text{sent}(P_i \to P_k, e_{mn})$$

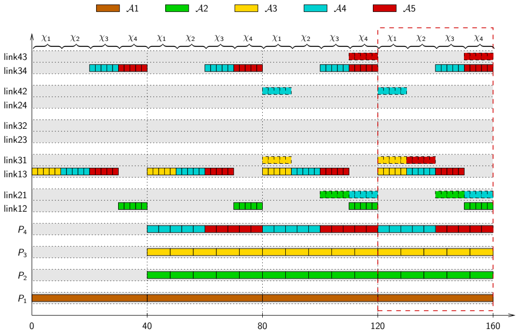
Computations

$$\sum_m \text{executed}(P_i, T_m) \times \text{flops}(T_m) \times w_i \leq 1$$

Outgoing communications

$$\sum_{m,n} \sum_j \text{sent}(P_j \to P_i, e_{mn}) \times \text{bytes}(e_{mn}) \times c_{ij} \leq 1$$

. . . but schedule reconstruction is harder



- ☺ Actual (cyclic) schedule obtained in polynomial time
- ☺ Asymptotic optimality
- ☹ A couple of practical problems (large period, # buffers)
- ☹ No **local** scheduling policy

The beauty of steady-state scheduling

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering of tasks/messages not needed
- Characterize resource activity per time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving from or sending to which neighbor?

Efficiency Optimal throughput $\Rightarrow$ optimal schedule (up to a constant number of tasks)

Periodic schedule, described in compact form
$\Rightarrow$ compiling a loop instead of a DAG!

Lesson learnt?

Resource selection is mandatory

. . . implementation may still be dynamic,
provided that static allocation is enforced by scheduler

Example: demand-driven assignment of enrolled workers

Scheduling multiple applications

- Large-scale platforms not likely to be exploited in dedicated mode/single application
- Investigate scenarios in which multiple applications are simultaneously executed on the platform
  $\Rightarrow$ competition for CPU and network resources

Target problem

- Large complex platform: several clusters and backbone links
- One (divisible load) application running on each cluster
- Which fraction of the job to delegate to other clusters?
- Applications have different communication-to-computation ratios
- How to ensure fair scheduling and good resource utilization?

Linear program

MINIMIZE $\min_k \left\{ \frac{\alpha_k}{\pi_k} \right\}$,

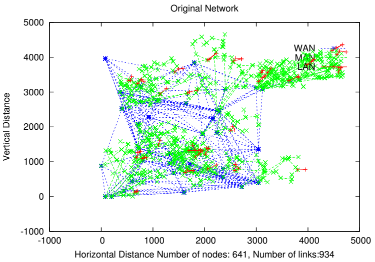
UNDER THE CONSTRAINTS

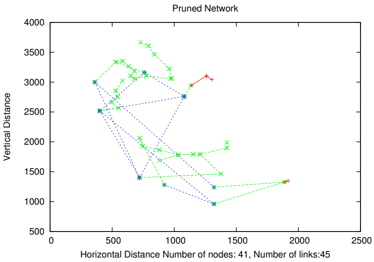
$$
\left\{
\begin{array}{lll}
(1a) & \forall C^k, & \sum_{l} \alpha_{k,l} = \alpha_k \\
(1b) & \forall C^k, & \sum_{l} \alpha_{l,k}.\tau_l \leq s_k \\
(1c) & \forall C^k, & \sum_{l \neq k} \alpha_{k,l}.\delta_k + \sum_{j \neq k} \alpha_{j,k}.\delta_j \leq g_k \\
(1d) & \forall l_i, & \sum_{l_i \in L_{k,l}} \beta_{k,l} \leq \text{max-connect}(l_i) \\
(1e) & \forall k,l, & \alpha_{k,l}.\delta_k \leq \beta_{k,l} \times g_{k,l} \\
(1f) & \forall k,l, & \alpha_{k,l} \geq 0 \\
(1g) & \forall k,l, & \beta_{k,l} \in \mathbb{N}
\end{array}
\right. \qquad (1)
$$

Approach

- Solution to *rational* linear problem as comparator/upper bound
- Several heuristics, greedy and LP-based
- Use Tiers as topology generator, and then SimGrid

Methodology (cont'd)





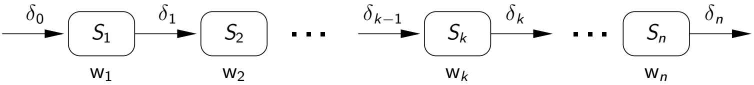
	distribution
K	$5, 7, \ldots, 90$
$\log(bw(l_k))$, $\log(g_k)$	normal ($mean=\log(2000)$, $std=\log(10)$)
s_k	uniform, 1000 — 10000
max-connect, δ_k, τ_k, π_k	uniform, 1 — 10

Platform parameters used in simulation

Hints for implementation

- Participants sharing resources in a Virtual Organization
- Centralized broker managing applications and resources
- Broker gathers all parameters of LP program
- Priority factors
- Various policies and refinements possible
  $\Rightarrow$ e.g. fixed number of connections per application

The application



- Consecutive data-sets fed into pipeline
- Period T_{period} = time interval between beginning of execution of two consecutive data sets
- Latency T_{latency} = time elapsed between beginning and end of execution for a given data set

Open problems

Single workflow

- Period/latency bi-criteria optimization
- Robust mappings
- Data-parallel stages (decreases latency)
- Replicated stages (decreases period & increases robustness)

Several (concurrent) workflows

- Competition for CPU and network resources
- Fairness between applications (max-min throughput, max stretch)
- Sensitivity to application/platform parameter changes

Lesson learnt?

Period, latency, stretch, robustness, fairness and combination lead to difficult optimization problems

Lot of work for young and talented algorithmicians ☺

Example: almost everything yet to be done!

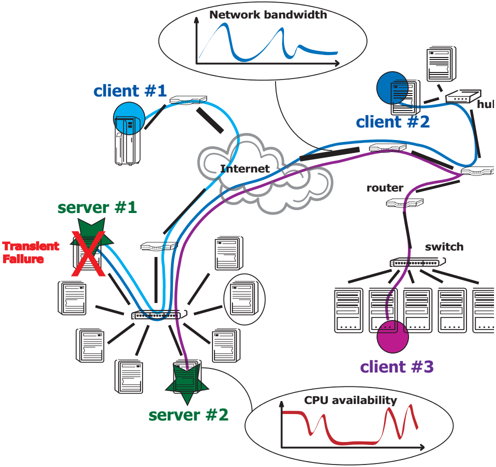
Knowledge of the platform graph

- For regular problems, the *structure* of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from *weights*, i.e. the estimation of execution and communication times
- Classical answer: *"use the past to predict the future"*
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
  $\Rightarrow$ This information guides scheduling decisions for next phase

Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- *Trace-based simulation*: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use SimGrid, an event-driven simulation toolkit

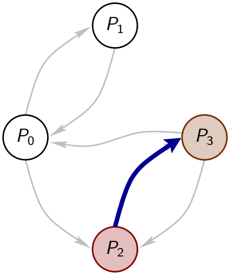
SimGrid traces

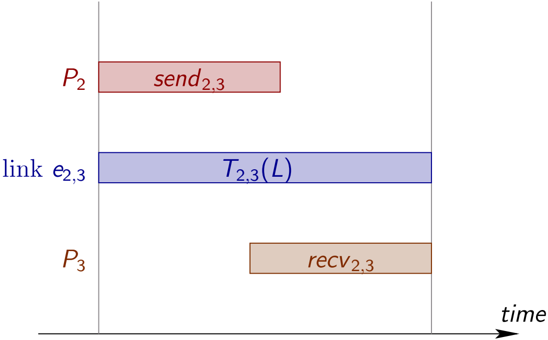


See http://simgrid.gforge.inria.fr/

Across physical links

Network = directed graph $\mathcal{P} = (V, E)$

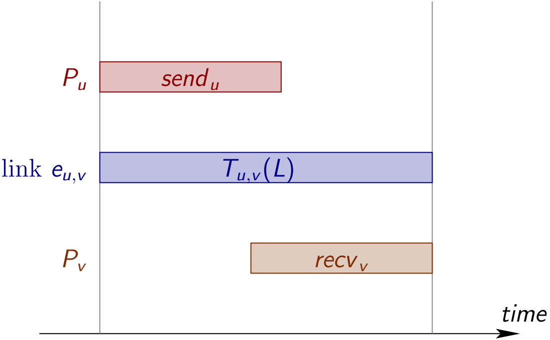




- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Multi-port

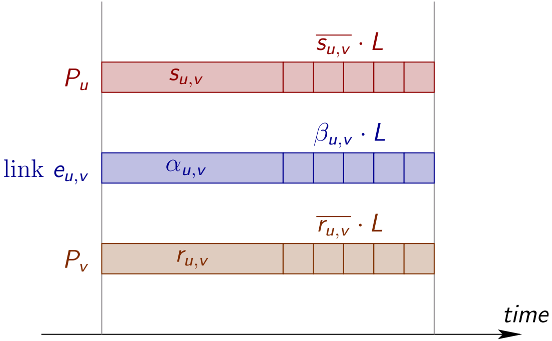
- Bar-Noy, Guha, Naor, Schieber:
  occupation time of sender P_u independent of target P_v



not *fully* multi-port model, but allows for starting a new transfer from P_u without waiting for previous one to finish

One-port

- Bhat, Raghavendra and Prasanna:
  same parameters for sender P_u, link $e_{u,v}$ and receiver P_v



two flavors:

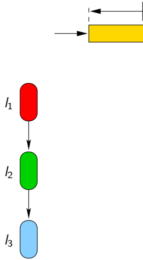
- bidirectional: simultaneous send *and* receive transfers allowed

- unidirectional: only one send or receive transfer at a given time-step

Store & Forward, WormHole, TCP

How to model a file transfer along a path?

$$\forall l \in Ł, \quad \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leq c_l$$



Max-Min Fairness maximize $\min_{r \in \mathcal{R}} \rho_r$

Proportional Fairness maximize $\sum_{r \in \mathcal{R}} \rho_r \log(\rho_r)$

MCT minimization maximize $\min_{r \in \mathcal{R}} \frac{1}{\rho_r}$

TCP behavior Close to max-min.

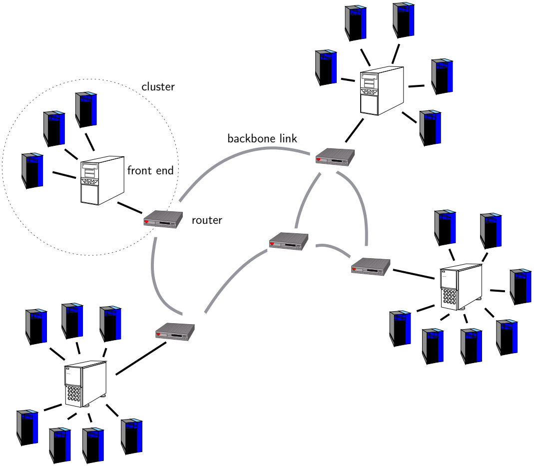
In SimGrid: max-min + bound by $1/RTT$

WormHolStore&pForwardAnalyticalmodel(packets),notthat realistic

Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow$ $bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

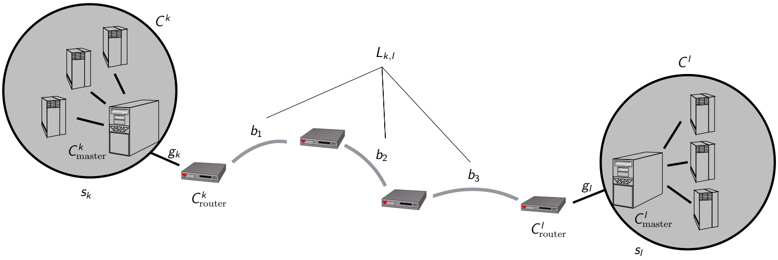
Sample large-scale platform



Accounts for Hierarchy + BW sharing

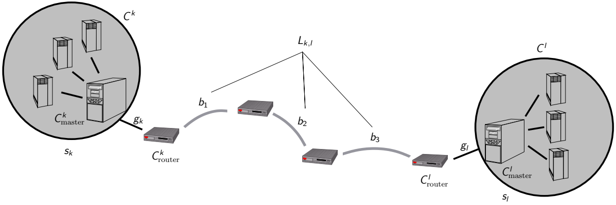
Assumes knowledge of Routing + Backbone bw + CPU speed

A first trial



Clusters and backbone links

A first trial (cont'd)



Clusters

- K clusters C^k, $1 \leq k \leq K$
- C^k_{master} front-end processor
- C^k_{router} router to external world
- s_k cumulated speed of C^k
- g_k bandwidth of the LAN link ($\gamma = 1$) from C^k_{master} to C^k_{router}

Network

- Set $\mathcal{R}$ of routers and $\mathcal{B}$ of backbone links l_i

How to cope with uncertainties and dynamicity? (1)

Sensibility analysis

- Asses the impact of uncertainties on existing solutions

Design robust solutions

- Robust optimization
  A robust solution remains "close" to optimal for all scenarios
- Internet-based computing
  No knowledge on task execution times
  Minimize risk taken while making any scheduling decision

How to cope with uncertainties and dynamicity? (2)

Stochastic models

1. What are the relevant stochastic models?
   Most characteristics remain to be studied and modeled
2. How can we use them?
   Chance-constrained programming?
   Other mathematical tools?

Tools for the road

- Forget absolute makespan minimization
- Resource selection mandatory
- Divisible load (fractional tasks)
- Single application: period / latency / power / robustness
- Several applications: max-min fairness, MAX stretch
- Linear programming: absolute bound to assess heuristics

Scheduling for large-scale platforms

- If platform is well identified and relatively stable, try to:
  (i) accurately model hierarchical structure
  (ii) design well-suited and robust scheduling algorithms

- If platform is not stable enough, or if it evolves too fast, dynamic schedulers are the only option

- Otherwise, grab any opportunity to

  inject static knowledge into dynamic schedulers

  ☹ Is this opportunity a niche?
  ☺ Does it encompass a wide range of applications?