On Cluster Resource Allocation for Multiple Parallel Task Graphs

Henri Casanova   Frédéric Desprez   Frédéric Suter

University of Hawai'i at Manoa

INRIA - LIP - ENS Lyon

IN2P3 Computing Center, CNRS / IN2P3

3rd Scheduling workshop
Aussois, France
June 2, 2010

Context

Scientific Workflow

- Application represented by a Directed Acyclic Graph (DAG)
  - Nodes $\Rightarrow$ Computations (usually sequential)
  - Edges $\Rightarrow$ Precedence constraints and communications

Evolution of Processor Architectures

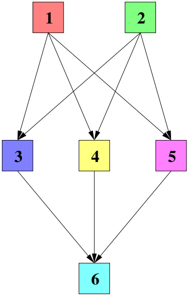
- Performance no longer coming from clock rate increase
  - Heat dissipation issues, high power consumption
- Multi- and Many-cores arising to keep pace with Moore's Law
- Memory becomes the new bottleneck
  - Risks of using 1 core per chip only for some applications

Question

- How to make workflows benefit from new processor architectures?

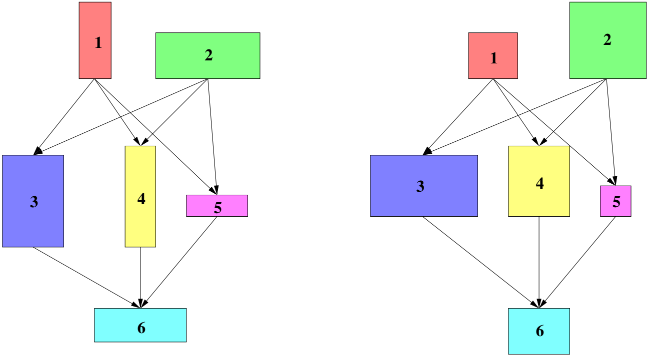
Next Generation Workflows?

From sequential ...



Next Generation Workflows?

... To moldable



- Advantages
  - Keep the task-parallelism of the workflow structure
  - Add data-parallelism in task execution
- Challenge
  - Good scheduling algorithms

How to schedule such Parallel Task Graphs?

Two steps

1. Determine the right number of processing units per node ⇒ Allocation
2. Find the "right" set of resources to execute each node ⇒ Mapping

Seminal algorithms

- CPA: Critical Path and Area-based scheduling
  - Radulescu and Van Gemund [ICPP 2001]
- Some variants: MCPA (Modified CPA), HCPA (Heterogeneous CPA), biCPA (bi-criteria CPA)

Objective functions

- Minimizing the completion time or the work needed
- Bi-criteria optimization

How to schedule such Parallel Task Graphs?

Two steps

1. Determine the right number of processing units per node ⇒ Allocation
2. Find the "right" set of resources to execute each node ⇒ Mapping

Seminal algorithms

- CPA: Critical Path and Area-based scheduling
  - Radulescu and Van Gemund [ICPP 2001]
- Some variants: MCPA (Modified CPA), HCPA (Heterogeneous CPA), biCPA (bi-criteria CPA)

Objective functions

- Minimizing the completion time or the work needed
- Bi-criteria optimization

Question

- How to schedule a batch of such PTGs?

Outline

- Introduction
- Notations and Performance Metrics
- The Naïve Solution: Be Selfish
- Some Alternatives
- Evaluation

Some Notations to Begin

- Simultaneous execution of N PTGs on a cluster of P processors
- Each PTG is a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
  - $\mathcal{V} = \{v_i \mid i = 1, \ldots, V\} \rightarrow$ data-parallel tasks
  - $\mathcal{E} = \{e_{i,j} \mid (i,j) \in \{1, \ldots, V\} \times \{1, \ldots, V\}\} \rightarrow$ precedence constraints
- No communication costs

- $T(v, p) \rightarrow$ execution time of task v on p processors
- $\omega(v) = T(v, p) \times p \rightarrow$ work relative to the execution of task v on p procs
- $bl(v) \rightarrow$ *bottom level* of task v

- $C^*_{max_i} \rightarrow$ makespan of the i^{th} PTG on the dedicated cluster
- $C_{max_i} \rightarrow$ makespan of the i^{th} PTG in the presence of competition

Performance Metrics

Average Stretch

- Average performance as perceived by the PTGs
- $\sum_{i=1}^{N} C_{max_i} \; / \; \sum_{i=1}^{N} C^*_{max_i}$.

Overall Makespan

- Standard metric for the performance of the whole batch
- $\max_{i=1,\dots,N} C_{max_i}$.

Maximum Stretch

- A measure of fairness
  - If optimally minimized $\Rightarrow$ PTGs have the same stretch $\Rightarrow$ fairness is optimal.
- $\max_{i=1,\dots,N} C^*_{max_i} / C_{max_i}$

Principle of CPA

Concept

- Find an allocation that is a good tradeoff between
  - Makespan: T_{CP}, the critical path length
  - Work: $T_A = \frac{1}{P}\sum_i W(v_i)$, the average area
  - While ($T_{CP} > T_A$)
    - One extra processor to the most critical task
- Mapping with a classical list scheduling heuristic

CPA's Allocation Procedure

1: **for all** $v \in \mathcal{V}$ **do**
2: $\quad p(v) \leftarrow 1$
3: **end for**
4: **while** $T_{CP} > T_A$ **do**
5: $\quad v \leftarrow$ task $\in$ CP $\mid \left(\frac{T(v,p(v))}{p(v)} - \frac{T(v,p(v)+1)}{p(v)+1}\right)$ is maximum
6: $\quad p(v) \leftarrow p(v) + 1$
7: $\quad$ **Update** T_A **and** T_{CP}
8: **end while**

The Naïve Solution: Be Selfish

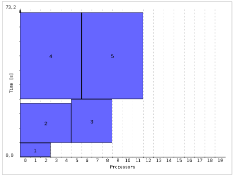
SELFISH

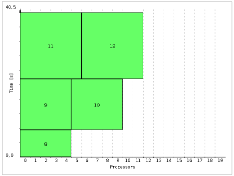
1. Compute an allocation for each PTG with CPA
   - Considering that the cluster is dedicated
2. Create a single scheduling list with all the tasks
3. Sort it by decreasing values of $bl(v)$
4. Map tasks to processors in order
5. Apply a conservative backfilling step

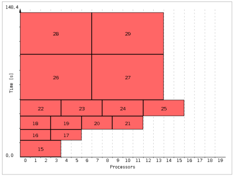
Potential Drawbacks

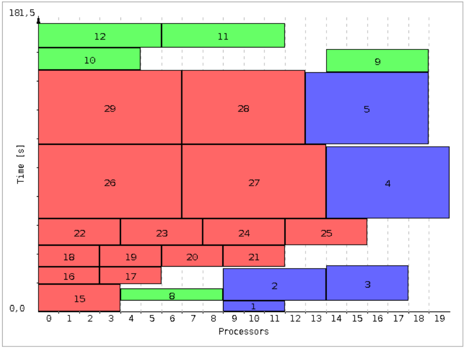
- Each PTG ignores the others → Concurrency
- No distinction between "short" and "long"
  - Tasks of short PTGs have small bottom level → Scheduled at last
  - ☹ Bad impact on fairness

Illustration









Makespan $= 181.5$ sec.

Fairness $= 1.82$

Outline

Improving the Mapping Step

Objective

- Give more importance to short PTG
  - Greatest impact on fairness

SELFISH_WEIGHT

- Sort tasks by decreasing $bl_{i,j} / \left(C^*_{max_i}\right)^2$

SELFISH_ORDER

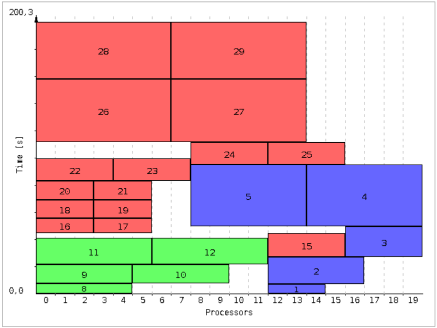
- Sort PTG by increasing $C^*_{max_i}$
- Then sort tasks by decreasing $bl_{i,j}$

Potential Drawbacks

☹ May increase the overall makespan

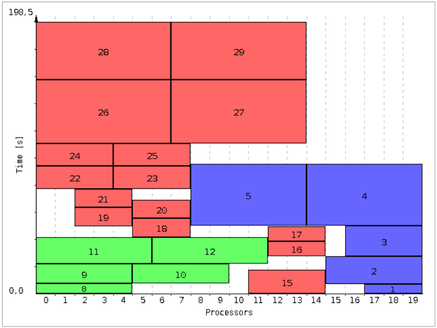
Illustration

SELFISH_WEIGHT



Makespan = 200.3 sec.
Fairness = 1.21

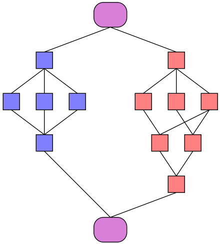
SELFISH_ORDER



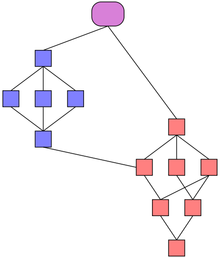
Makespan = 198.5 sec.
Fairness = 1.20

Combining the Graphs

- Proposed by Zhao and Sakellariou
  - For regular DAGS
- Merge all PTGs into one
- Then apply an algorithm for single PTG (i.e., CPA)



C1



C4

Potential Drawbacks

- ☹ C1 postpones the small PTGs → bad impact on fairness
- ☹ High level of concurrency badly handled by CPA

Distributing the Resources

Basic idea

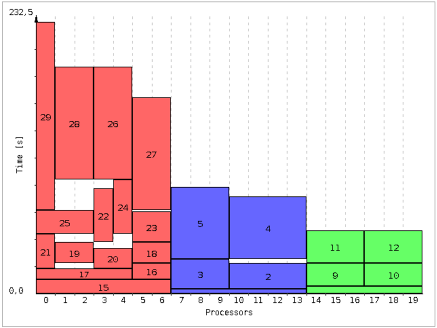
- Constrain each PTG in the allocation phase
  - Apply CPA as if the cluster has less processors, i.e., $P_i < P \mid \sum_{i=1}^{N} P_i = P$
- Different static constraints proportional to
  - The number of PTGs: $P_i = 1/N \rightarrow$ `CRA_NDAGS`
  - The work of each PTG: $P_i = \frac{1}{2N} + \frac{\omega_i}{2\sum_{j=1}^{N} \omega_j} \rightarrow$ `CRA_WORK`
  - The width of each PTG: $P_i = \frac{1}{2N} + \frac{width(i)}{2\sum_{j=1}^{N} width(j)} \rightarrow$ `CRA_WIDTH`
- Can be combined to `_WEIGHT` and `_ORDER` optimized mappings

Potential Drawbacks

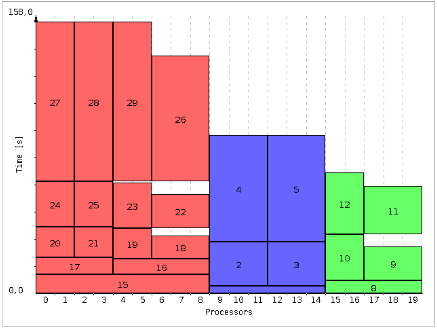
☹ Static constraints cannot account for
  - A PTG completion
  - Changes in shape

Illustration

CRA_NDAGS



CRA_WORK



- Constraints
  - $\mathrm{PTG}_0 = 7$
  - $\mathrm{PTG}_1 = 6$
  - $\mathrm{PTG}_2 = 7$

- Constraints
  - $\mathrm{PTG}_0 = 6$
  - $\mathrm{PTG}_1 = 5$
  - $\mathrm{PTG}_2 = 9$

What About Dynamic Constraints?

The MAGS Algorithm

- Consider PTGs as malleable jobs
- Sketch of the algorithm
  1. Determine scheduling periods
  2. Find an allocation for each PTG in each period
     2.1 It defines a malleable allocation
  3. Schedule the malleable allocations
- Malleable Allocations with Guaranteed Stretch
  - This algorithm even comes with a guarantee!

Determining the Scheduling Periods

- Start with a perfectly fair schedule
  - In which all PTGs experience the same stretch S
- Compute a lower bound of S called S^*
  - Assume that PTGs are ideally malleable jobs
  - The ith job should finish exactly at time $S \times C^*_{max_i}$
    - And all job j for $1 \leq j \leq i$ should finish before $S \times C^*_{max_i}$
    - Otherwise the ith job has a stretch greater than S
  - The sum of the works is lower than $P \times S \times C^*_{max_i}$

$$\forall i = 1, \ldots, n \quad \sum_{j=1}^{i} P \times C^*_{max_j} \leq P \times S \times C^*_{max_i}$$

  - The lower bound S^* on the stretch is then

$$S^* = \max_{i=1,\ldots,n} \frac{1}{C^*_{max_i}} \sum_{j=1}^{i} C^*_{max_j}$$

- This leads to N periods finishing at $S^* \times C^*_{max_i}$
  - ☹ Many periods
  - ☹ Some may be very small (too small to execute a single task)

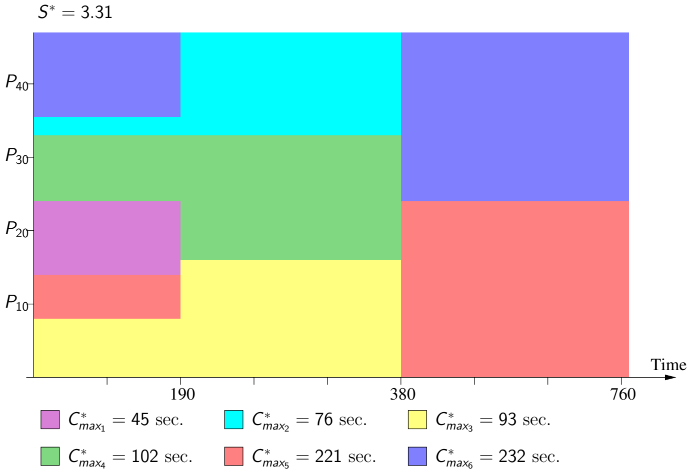
Relaxing the Perfectly Fair Schedule

- Structure the schedule in M periods
  - Period i lasts from t_{i-1} to t_i
  - $t_0 = 0$, the rest has to be be determined
- Job j finishes in period i_j in the perfectly fair schedule

  $t_{i_j - 1} \leq S^* \times C^*_{max_i} < t_{i_j}$
- Set $t_1 = S^* \times C_{max_1}$
  - Only job 1 may complete during the first period
- Use geometrically increasing periods
  - Define $t_{i+1} = t_i \times (1 + \lambda)$ for $i = 2, \ldots, M$ and some $\lambda > 0$
  - Then $t_i = S^* \times C_{max_1} \times (1 + \lambda)^{i-1}$ for $i = 1, \ldots, N$
- The stretch of each job is smaller than $(1 + \lambda)S^*$
- $\lambda = \max(1, \pi/(S^* C^*_{max_1}))$
  - With π the smallest allowed period

☺ Guarantee that no job is more than a factor 2 away from S^*

Illustration



Scheduling PTGs in Malleable Allocations

- Objective: finish no later than $S^* \times C^*_{max_i}$
- Consider the periods in reverse order
- Schedule the tasks in a bottom-up fashion
  - From the exit tasks towards the entry task
- Why?
  - The exit task of each PTG finishes exactly at the end of its last period
- Use CPA to determine the allocation of each task
  - As if the cluster comprised the alloted number of processors in the current period

Relaxing the Stretch Guarantee

- Jobs aren't perfectly malleable
  - Some tasks may not complete before the end of a period
  - Postponing would be bad!
- Solution: introduce some slack
- The guarantee is now $slack \times 2 \times S^*$
  - Where $slack \geq 1$
- How to find the smallest slack leading to a feasible schedule?
  1. Start with $slack = 1$
  2. Double the value of the slack until a schedule is found
  3. Apply a binary search

Consider PTGs as Independent Moldable Jobs

$3/2 + \epsilon$ approximation algorithm

- Proposed by Dutot `et al.` In the Handbook of Scheduling, Chapter 26
  - Computes an approximation of the optimal makespan C^*_{max}
  - Computes an allocation for each job
  - Schedules in two shelves
    - Larger jobs in the first shelf, Smaller jobs in the second one

Extension at SPAA'04

- Two different list scheduling strategies: `LPTF` and `SAF`
- A K-shelves approach
  - Partition the time in K phases, or "shelves"
    - Depends on C^*_{max} and the smallest execution time
  - For each shelf
    - Solve a knapsack problem to maximize the work executed in the current shelf

Adapting to our Context

- PTGs are a special kind of moldable jobs
  - Have to consider their fine grain structure too
- Our proposition $\rightarrow$ Coarse-grain Allocation and Fine-grain Mapping

The CAFM Approach

Global Sketch

1. Get a moldable profile for each job
   - Determine the makespan for each job for each number of processors
2. Determine a coarse-grain allocation for each moldable job
3. Schedule the "boxes" representing each selected job
4. Schedule each task graph within its box (fine-grain mapping)
5. Open the boxes
6. Do some backfilling

The variants

- CAFM_LPTF
- CAFM_SPTF
- CAFM_SAF
- CAFM_K_SHELVES
- CAFM_CRA[_WEIGHT|_ORDER]
  - skip step 3 and swap steps 4 and 5

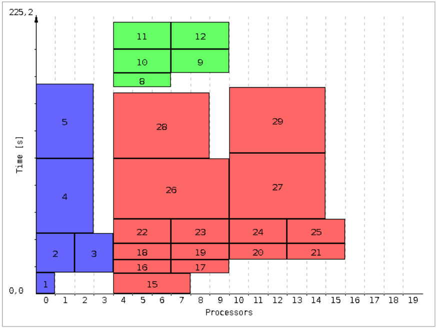
Potential Drawbacks

☹ Step 1 is really time-consuming

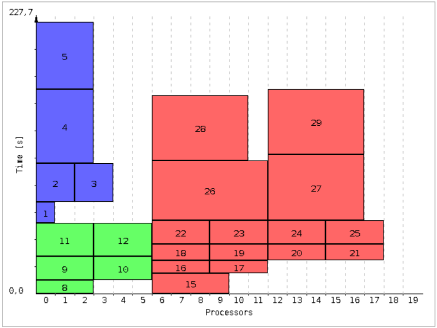
☹ LPTF favors long PTGs → bad for fairness

Illustration

CAFM_LPTF



CAFM_SPTF



- Moldable Allocations
  - $PTG_0 = 4$
  - $PTG_1 = 6$
  - $PTG_2 = 14$
- Before backfilling step

Outline

- Introduction
- Notations and Performance Metrics
- The Naïve Solution: Be Selfish
- Some Alternatives
- Evaluation

Experimental Settings

- Evaluation through simulation
  - SimGrid Toolkit v3.3
- Platforms
  - Three clusters of the Grid'5000 platform

Cluster	chti	grillon	grelon	gdx
#proc.	20	47	120	216
Gflop/sec.	4.311	3.379	3.185	3.388

  - Gigabit switched interconnect
    - $100\mu sec$ latency and $1Gb/sec$ bandwidth)
- Applications
  - Random PTGs (10, 20 or 30 tasks)
  - FFT-shaped PTGS (5, 15, or 39 tasks)
  - Strassen matrix multiplication (25 tasks)
  - No inter-task communication costs
  - Batches of 2, 4, 6, 8, and 10 PTGs
- Contenders
  - `SELFISH_*`, `CRA_*`, `CAFM_*` and `MAGS`

And the Winner is . . .

And the Winner is . . .

MAGS!!

If You Want More Details

Papers

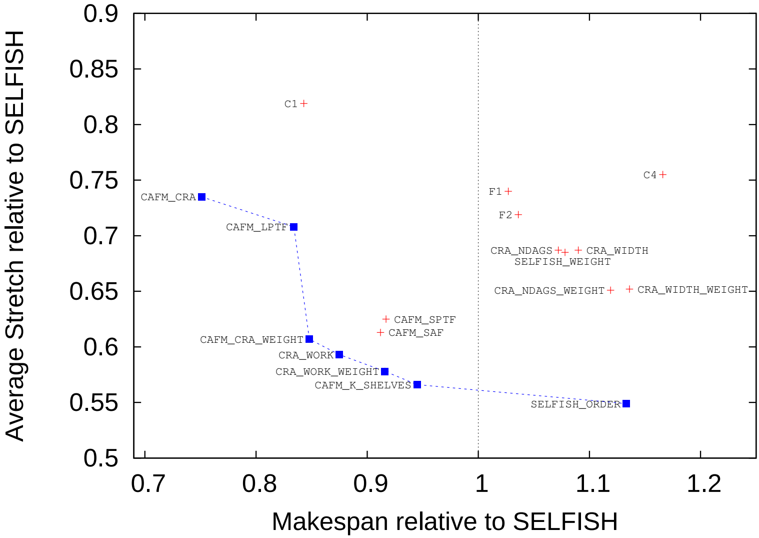
Henri Casanova, Frédéric Desprez and Frédéric Suter. On Cluster Resource Allocation for Multiple Parallel Task Graphs. Submitted to *Journal of Parallel and Distributed Computing*. Also available as INRIA Research Report RR-7224.

Henri Casanova, Frédéric Desprez and Frédéric Suter. Minimizing Stretch and Makespan of Multiple Parallel Task Graphs via Malleable Allocations. In *39th International Conference on Parallel Processing (ICPP 2010)*, San Diego, California, Sep 2010.

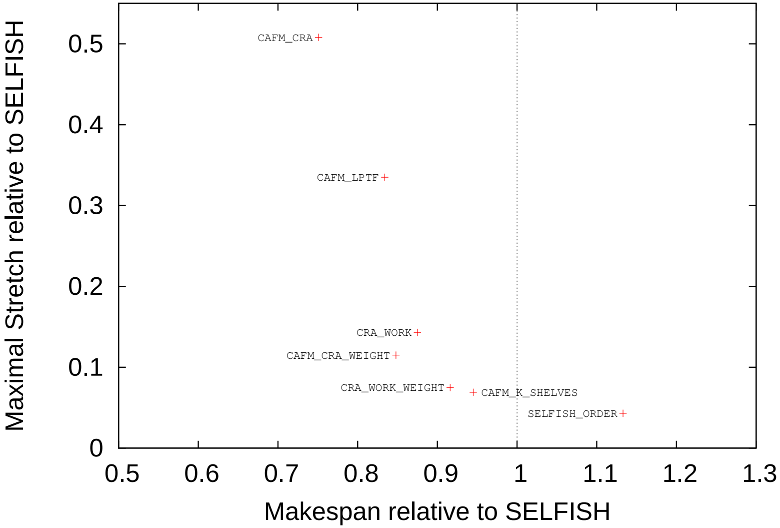
Tools

- DAGs generated with daggen
  - `http://www.loria.fr/~suter/dags.html`
- Output visualization with Jedule
  - `http://www.icsi.berkeley.edu/~sascha/jedule/index.html`

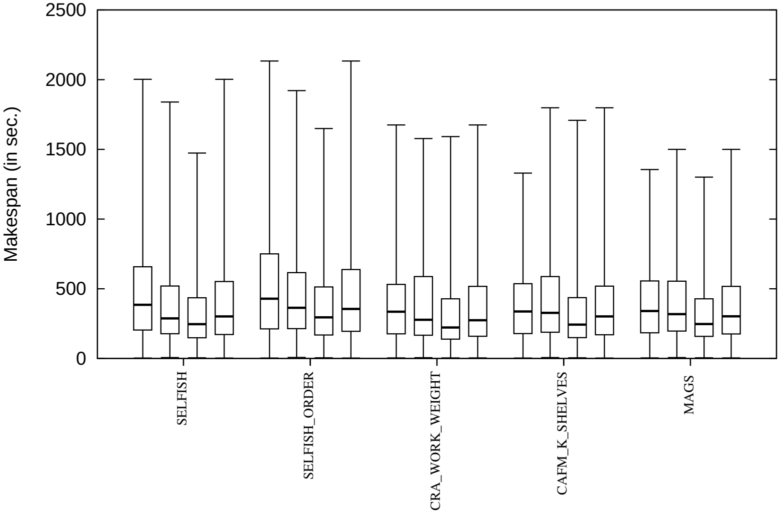
Makespan vs. Average Stretch



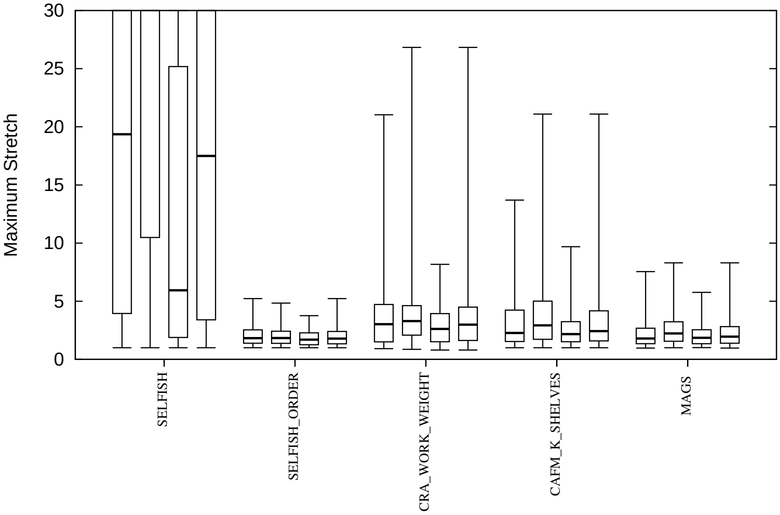
Makespan vs. Maximum Stretch



Makespan distribution



Maximum Stretch Distribution



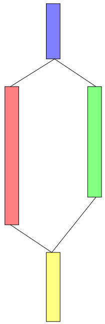
Contenders performance wrt MAGS

	Makespan	Average Stretch	Maximum Stretch
SELFISH	7%	75.97%	1909.54%
SELFISH_ORDER	21.27%	−3.42%	−13.68%
CRA_WORK_WEIGHT	−1.99%	1.77%	49.79%
CAFM_K_SHELVES	1.14%	−0.44%	38.54%

CPA on a small example: 4 tasks on 4 processors

Task	Execution Time			
T_1	**4**	2	1.5	1.5
T_2	**10**	6	4	3
T_3	**8**	5	3.5	3
T_4	**5**	3	2	1.5

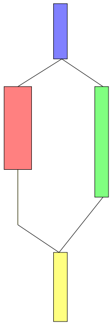
- $T_{CP} = 19$
- $T_A = 6.75$



CPA on a small example: 4 tasks on 4 processors

Task	Execution Time			
T_1	**4**	2	1.5	1.5
T_2	10	**6**	4	3
T_3	**8**	5	3.5	3
T_4	**5**	3	2	1.5

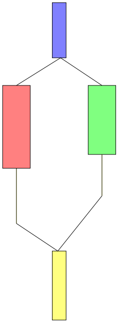
- $T_{CP} = 17$
- $T_A = 7$



CPA on a small example: 4 tasks on 4 processors

Task	Execution Time			
T_1	**4**	2	1.5	1.5
T_2	10	**6**	4	3
T_3	8	**5**	3.5	3
T_4	**5**	3	2	1.5

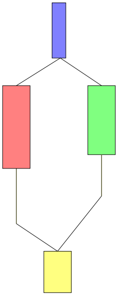
- $T_{CP} = 15$
- $T_A = 7.75$



CPA on a small example: 4 tasks on 4 processors

Task	Execution Time			
T_1	**4**	2	1.5	1.5
T_2	10	**6**	4	3
T_3	8	**5**	3.5	3
T_4	5	**3**	2	1.5

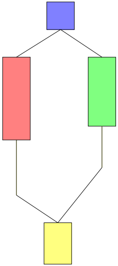
- $T_{CP} = 13$
- $T_A = 8$



CPA on a small example: 4 tasks on 4 processors

Task	Execution Time			
T_1	4	**2**	1.5	1.5
T_2	10	**6**	4	3
T_3	8	**5**	3.5	3
T_4	5	**3**	2	1.5

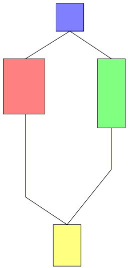
- $T_{CP} = 11$
- $T_A = 8$



CPA on a small example: 4 tasks on 4 processors

Task	Execution Time			
T_1	4	**2**	1.5	1.5
T_2	10	6	**4**	3
T_3	8	**5**	3.5	3
T_4	5	**3**	2	1.5

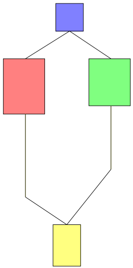
- $T_{CP} = 10$
- $T_A = 8$



CPA on a small example: 4 tasks on 4 processors

Task	Execution Time			
T_1	4	**2**	1.5	1.5
T_2	10	6	**4**	3
T_3	8	5	**3.5**	3
T_4	5	**3**	2	1.5

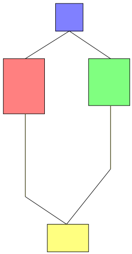
- $T_{CP} = 9$
- $T_A = 8.125$



CPA on a small example: 4 tasks on 4 processors

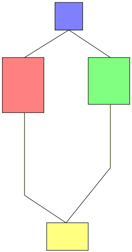
Task	Execution Time			
T_1	4	**2**	1.5	1.5
T_2	10	6	**4**	3
T_3	8	5	**3.5**	3
T_4	5	3	**2**	1.5

- $T_{CP} = 8$
- $T_A = 8.125$



CPA on a small example: 4 tasks on 4 processors

Task	Execution Time			
T_1	4	**2**	1.5	1.5
T_2	10	6	**4**	3
T_3	8	5	**3.5**	3
T_4	5	3	**2**	1.5



- $T_{CP} = 8$
- $T_A = 8.125$