Co-scheduling algorithms for high-throughput workload execution

Guillaume Aupy[1], Manu Shantharam[2], Anne Benoit[1,3], Yves Robert[1,3,4] and Padma Raghavan[5]

1. Ecole Normale Supérieure de Lyon, France
2. University of Utah, USA
3. Institut Universitaire de France
4. University of Tennessee Knoxville, USA
5. Pennsylvania State University, USA

Anne.Benoit@ens-lyon.fr http://graal.ens-lyon.fr/~abenoit/

9th Scheduling for Large Scale Systems Workshop
July 1-4, 2014 - Lyon, France

Motivation

- Execution time of HPC applications
  - Can be significantly reduced when using a large number of processors
  - But inefficient resource usage if all resources used for a single application (non-linear decrease of execution time)

- Pool of several applications
  - Co-scheduling algorithms: execute several applications concurrently
  - Increase individual execution time of each application, but
    - (i) improve efficiency of parallelization
    - (ii) reduce total execution time
    - (iii) reduce average response time

- Increase platform yield, and save energy

Framework

- Distributed-memory platform with p identical processors
- Set of n independent tasks (or applications) $T_1, \ldots, T_n$; application T_i can be assigned $\sigma(i) = j$ processors, and
  - p_i is the minimum number of processors required by T_i;
  - $t_{i,j}$ is the execution time of task T_i with j processors;
  - $work(i,j) = j \times t_{i,j}$ is the corresponding work.
- We assume the following for $1 \leq i \leq n$ and $p_i \leq j < p$:

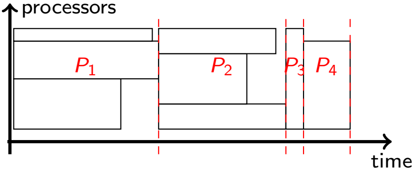
Non increasing execution time: $t_{i,j+1} \leq t_{i,j}$

Non decreasing work: $work(i,j+1) \geq work(i,j)$

Co-schedules

A co-schedule partitions the n tasks into groups (called *packs*):

- All tasks from a given pack start their execution at the same time
- Two tasks from different packs have disjoint execution intervals



A co-schedule with four packs P_1 to P_4

Definition (k-IN-p-COSCHEDULE optimization problem)

Given a fixed constant $k \leq p$, find a co-schedule with at most k tasks per pack that minimizes the execution time.

The most general problem is when $k = p$, but in some frameworks we may have an upper bound $k < p$ on the maximum number of tasks within each pack.

Related work

- *Performance bounds for level-oriented two-dimensional packing algorithms*, Coffman, Garey, Johnson:
  Strip-packing problem, parallel tasks (fixed number of processors), approximation algorithm based on "shelves"
- *Scheduling parallel tasks: Approximation algorithms*, Dutot, Mounié, Trystram:
  Use this model to approximate the moldable model; they studied the p-IN-p-COSCHEDULE for identical moldable tasks (polynomial with DP)
- Widely studied for sequential tasks

Complexity: Polynomial instances

Theorem

The 1-IN-*p*-COSCHEDULE *and* 2-IN-*p*-COSCHEDULE *problems can both be solved in polynomial time.*

Proof.

If there is a batch with exactly tasks T_i and $T_{i'}$, then its execution time is $\min_{j=p_i..p-p_{i'}}\left(\max(t_{i,j}, t_{i',p-j})\right)$.

We then construct the complete weighted graph $G = (V, E)$, where $|V| = n$, and

$$e_{i,i'} = \begin{cases} t_{i,p} & \text{if } i = i' \\ \min_{j=p_i..p-p_{i'}}\left(\max(t_{i,j}, t_{i',p-j})\right) & \text{otherwise} \end{cases}$$

Finally, finding a perfect matching of minimal weight in G leads to the optimal solution for 2-IN-*p*-COSCHEDULE. □

Complexity: Polynomial instances

Theorem

The 1-IN-p-COSCHEDULE *and* 2-IN-p-COSCHEDULE *problems can both be solved in polynomial time.*

Proof.

If there is a batch with exactly tasks T_i and $T_{i'}$, then its execution time is $\min_{j=p_i..p-p_{i'}} \left(\max(t_{i,j}, t_{i',p-j})\right)$.

We then construct the complete weighted graph $G = (V, E)$, where $|V| = n$, and

$$e_{i,i'} = \begin{cases} t_{i,p} & \text{if } i = i' \\ \min_{j=p_i..p-p_{i'}} \left(\max(t_{i,j}, t_{i',p-j})\right) & \text{otherwise} \end{cases}$$

Finally, finding a perfect matching of minimal weight in G leads to the optimal solution for 2-IN-p-COSCHEDULE. □

Complexity: NP-completeness

Theorem

The 3-IN-p-COSCHEDULE problem is strongly NP-complete.

Proof.

We reduce this problem to 3-PARTITION: Given an integer B and $3n$ integers $a_1, \ldots, a_{3n}$, can we partition the $3n$ integers into n triplets, each of sum B? This problem is strongly NP-hard so we can encode the a_i's and B in unary.

We build instance $\mathcal{I}_2$ of 3-IN-p-COSCHEDULE, with $p = B$ processors, a deadline $D = n$, and $3n$ tasks T_i such that $t_{i,j} = 1 + \frac{1}{a_i}$ if $j < a_i$, $t_{i,j} = 1$ otherwise. *(The $t_{i,j}$'s verify the constraints on work and execution time.)*

Any solution of $\mathcal{I}_2$ has n packs each of cost 1 with exactly 3 tasks in it, and the sum of the weights of these tasks sums up to B. □

Complexity: NP-completeness

Theorem

The 3-IN-p-COSCHEDULE problem is strongly NP-complete.

Proof.

We reduce this problem to 3-PARTITION: Given an integer B and $3n$ integers $a_1, \dots, a_{3n}$, can we partition the $3n$ integers into n triplets, each of sum B? This problem is strongly NP-hard so we can encode the a_i's and B in unary.

We build instance $\mathcal{I}_2$ of 3-IN-p-COSCHEDULE, with $p = B$ processors, a deadline $D = n$, and $3n$ tasks T_i such that $t_{i,j} = 1 + \frac{1}{a_i}$ if $j < a_i$, $t_{i,j} = 1$ otherwise. *(The $t_{i,j}$'s verify the constraints on work and execution time.)*

Any solution of $\mathcal{I}_2$ has n packs each of cost 1 with exactly 3 tasks in it, and the sum of the weights of these tasks sums up to B. □

Complexity: NP-completeness

Theorem

For $k \geq 3$, The k-IN-p-COSCHEDULE problem is strongly NP-complete.

Proof.

We reduce these problems to the same instance of the 3-IN-p-COSCHEDULE problem, to which we further add:

- $n(k-3)$ buffer tasks such that $t_{i,j} = \max\left(\frac{B+1}{j}, 1\right)$;
- the number of processors is now $p = B + (k-3)(B+1)$;
- the deadline remains $D = n$.

Again, we need to execute each pack in unit time and at most n packs. The only way to proceed is to execute within each pack $k-3$ buffer tasks on $B+1$ processors. □

Complexity: NP-completeness

Theorem

For $k \geq 3$, The k-IN-p-COSCHEDULE problem is strongly NP-complete.

Proof.

We reduce these problems to the same instance of the 3-IN-p-COSCHEDULE problem, to which we further add:

- $n(k-3)$ buffer tasks such that $t_{i,j} = \max\left(\frac{B+1}{j}, 1\right)$;
- the number of processors is now $p = B + (k-3)(B+1)$;
- the deadline remains $D = n$.

Again, we need to execute each pack in unit time and at most n packs. The only way to proceed is to execute within each pack $k-3$ buffer tasks on $B+1$ processors. □

Scheduling a pack of tasks

Theorem

Given k tasks to be scheduled on p processors in a single pack (1-pack-schedule), we can find in time $O(p \log k)$ the schedule that minimizes the cost of the pack.

Greedy algorithm Optimal-1-pack-schedule:

- Initially, each task T_i is assigned its minimum number of processors p_i
- While there remain available processors, assign one to the largest task (with their current processor assignment)

This algorithm returns an optimal solution

Scheduling a pack of tasks

Theorem

Given k tasks to be scheduled on p processors in a single pack (1-pack-schedule), we can find in time $O(p \log k)$ the schedule that minimizes the cost of the pack.

Greedy algorithm Optimal-1-pack-schedule:

- Initially, each task T_i is assigned its minimum number of processors p_i
- While there remain available processors, assign one to the largest task (with their current processor assignment)

This algorithm returns an optimal solution

Optimal solution

Theorem

The following integer linear program characterizes the k-IN-p-COSCHEDULE problem, where the unknown variables are the $x_{i,j,b}$'s (Boolean variables) and the y_b's (rational variables), for $1 \leq i, b \leq n$ and $1 \leq j \leq p$:

$$
\begin{array}{ll}
\text{Minimize} \sum_{b=1}^{n} y_b & \text{subject to} \\
\text{(i)} \ \sum_{j,b} x_{i,j,b} = 1, & 1 \leq i \leq n \\
\text{(ii)} \ \sum_{i,j} x_{i,j,b} \leq k, & 1 \leq b \leq n \\
\text{(iii)} \ \sum_{i,j} j \times x_{i,j,b} \leq p, & 1 \leq b \leq n \\
\text{(iv)} \ x_{i,j,b} \times t_{i,j} \leq y_b, & 1 \leq i, b \leq n, 1 \leq j \leq p
\end{array}
$$

$x_{i,j,b} = 1$ iff T_i is in pack b and executed on j processors

y_b is the execution time of pack b

Approximation algorithm

- 3-approximation algorithm for the problem p-IN-p-COSCHEDULE
- Initialization: task T_i executed on p_i processors
- Greedy procedure MAKE-PACK to create packs (with $k = p$), given $\sigma(i)$ processors for task T_i

procedure MAKE-PACK(n, p, k, σ)
begin
  L: list of tasks sorted in non-increasing execution times $t_{i,\sigma(i)}$;
  while $L \neq \emptyset$ **do**
    Schedule the current task on the first pack with enough available processors and less than k tasks;
    Create a new pack if no existing pack fits;
    Remove the current task from L;
  end
  return *the set of packs*
end

- PACK-APPROX: Iteratively refine the solution, adding a processor to the task with longest execution time

```
procedure PACK-APPROX(T_1, ..., T_n)
begin
    COST = +∞;
    for j = 1 to n do  σ(j) ← p_j ;
    for i = 0 to Σ_j (p − p_j) − 1 do
        Call MAKE-PACK (n, p, p, σ);
        Let COST_i be the cost of the co-schedule;
        if COST_i < COST then  COST ← COST_i;
        Let A_tot(i) = Σ_{j=1}^{n} t_{j,σ(j)} σ(j);
        Let T_{j*} be one task that maximizes t_{j,σ(j)};
        if (A_tot(i) > p × t_{j*,σ(j*)}) or (σ(j*) = p) then
            return COST
        else
            σ(j*) ← σ(j*) + 1
        end
    end
    return COST;
end
```

Theorem

PACK-APPROX *is a 3-approximation algorithm for the* p-IN-p-COSCHEDULE *problem.*

Involved proof, studying the different ways to exit algorithm PACK-APPROX:

- The task with longest execution time is already assigned p processors
- The sum of the work of all tasks $(\sum_{i=1}^{n} t_{i,\sigma(i)}\sigma(i))$ is greater than p times the longest execution time
- Each task has been assigned p processors

Heuristics

In all heuristics (even randoms), once the different packs are chosen, we always run Optimal-1-pack-schedule *on each pack.*

RANDOM-PACK: generates the packs randomly: randomly chooses an integer j between 1 and k, and then randomly selects j tasks to form a pack.

RANDOM-PROC: assigns the number of processors to each task randomly, then calls MAKE-PACK to generate the packs.

PACK-BY-PACK (ε): creates packs that are "well-balanced": the difference between smallest and longest execution times of a pack is small (ratio of $1 + \varepsilon$).

PACK-APPROX: an extension of the approximation algorithm in the case where there are at most k tasks in a pack.

Heuristic variants

Improvement of the heuristics by using up to 9 runs:

- 4 *random* heuristics with either one or nine runs:
  - Random-Pack-1, Random-Pack-9
  - Random-Proc-1, Random-Proc-9
- pack-by-pack (ε) with
  - either one single run with $\varepsilon = 0.5$ (pack-by-pack-1)
  - or 9 runs with $\varepsilon \in \{.1, .2, \ldots, .9\}$ (pack-by-pack-9)
- Only one version of pack-Approx

Further variants: up to 99 runs, or better choice to create packs in pack-by-pack, but only little improvement at the price of a much higher running time

Heuristic variants

Improvement of the heuristics by using up to 9 runs:

- 4 *random* heuristics with either one or nine runs:
  - Random-Pack-1, Random-Pack-9
  - Random-Proc-1, Random-Proc-9
- pack-by-pack (ε) with
  - either one single run with $\varepsilon = 0.5$ (pack-by-pack-1)
  - or 9 runs with $\varepsilon \in \{.1, .2, \ldots, .9\}$ (pack-by-pack-9)
- Only one version of pack-Approx

Further variants: up to 99 runs, or better choice to create packs in pack-by-pack, but only little improvement at the price of a much higher running time

Workloads

- Workload-I: 10 parallel scientific applications (involving VASP, ABAQUS, LAMMPS, Petsc); execution time observed on a cluster with $p = 16$ processors and 128 cores
- Workload-II: synthetic test suite with 65 tasks for 128 cores ($p = 16$); execution time for problem size m on q cores:

$$t(m,q) = f \times t(m,1) + (1-f)\frac{t(m,1)}{q} + \kappa(m,q)$$

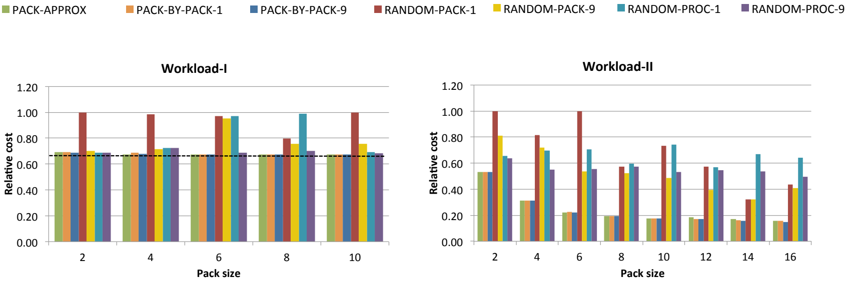
  - f: inherently serial fraction
  - κ: overheads related to synchronization and communication
- Workload-III: similar to Workload-II, but with 260 tasks for 256 cores ($p = 32$)

Assessing the performance of heuristics

- Seven heuristics and three measures:

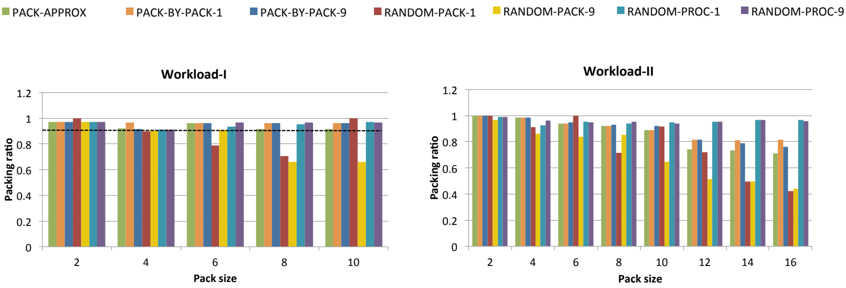
- Relative cost: cost divided by the cost of a schedule with each task scheduled on p processors (schedule used in practice, n-packs-schedule)
- Packing ratio: total work $\sum_{i=1}^{n} t_{i,\sigma(i)} \times \sigma(i)$ divided by p times the cost of the co-schedule; close to 1 if no idle time
- Relative response time: mean response time compared to n-packs-schedule with non-decreasing order of execution time

Results: Relative cost



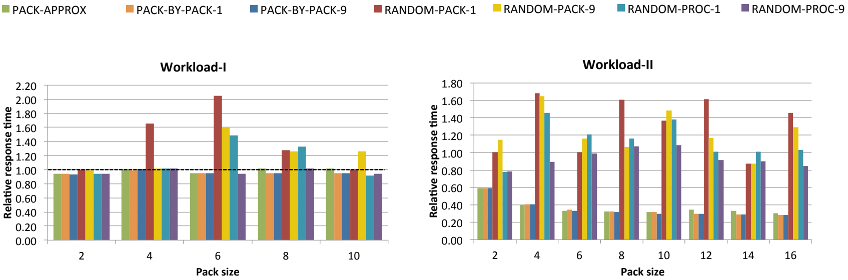
- Horizontal line = optimal co-schedule (exhaustive search for W-I)
- PACK-APPROX and PACK-BY-PACK close to optimal
- Gain of more than 35% compared to *n*-packs-schedule for W-I
- Huge gains for W-II (more than 80%, better for larger values of pack size)

Results: Packing ratio



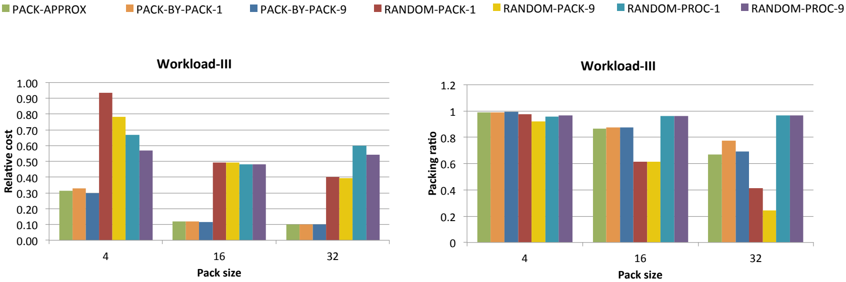
- Packing ratios very close to one for PACK-BY-PACK and PACK-APPROX
- High quality packings

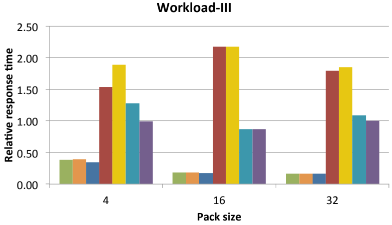
Results: Response time



- Values less than 1: improvements in response times
- For Workload-II and larger values of the pack size, response time gains over 80%
- *k*-IN-*p*-COSCHEDULE attractive from the user perspective

Results: Workload-III





- Scalability trends with 260 tasks on 32 processors
- PACK-APPROX and PACK-BY-PACK are clearly superior

Results: Running times

	Workload-I	Workload-II	Workload-III
PACK-APPROX	0.50	0.30	5.12
PACK-BY-PACK-1	0.03	0.12	0.53
PACK-BY-PACK-9	0.30	1.17	5.07
RANDOM-PACK-1	0.07	0.34	9.30
RANDOM-PACK-9	0.67	2.71	87.25
RANDOM-PROC-1	0.05	0.26	4.49
RANDOM-PROC-9	0.47	2.26	39.54

- Average running times in milliseconds
- All heuristics run within a few ms, even for W-III
- Random heuristics slower (cost of random number generation)
- PACK-BY-PACK-9 comparable with PACK-APPROX

Conclusion

- *Theoretically:* Exhaustive complexity study
  - NP-completeness (need to choose for each task both number of processors and pack)
  - Optimal strategy once the packs are formed
  - Efficient algorithm to partition tasks with pre-assigned resources into packs (3-approximation algorithm for $k = p$)
- *Practically:* Heuristics building upon theoretical study, with very good performance
  - Heuristic of choice: PACK-BY-PACK-9
  - Great improvement compared to existing schedulers (in terms of relative cost)
  - Corresponding savings in system energy cost
  - Measurable benefits in average response time

Future work

- Combine with DVFS technique (dynamic voltage and frequency scaling) to further obtain gains in energy consumption
- Experiment at a larger scale (university computing facilities), where workload attributes do not vary much in time, and energy costs are a limiting factor
- Theoretically, obtain more approximation results