Decentralized List Scheduling

Marc Tchiboukdjian Denis Trystram

Laboratoire d'Informatique de Grenoble

INRIA

Parallel Programming

Parallel platforms

- Multicores
- Manycores (GPU)

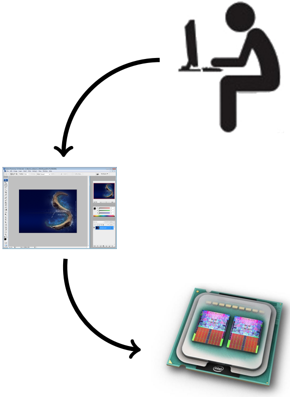
Characteristics

- Increasing number of cores
- Shared memory

Parallel Programming

Need for parallel algorithms that are

- easy to program
- efficient even on small data sets



Task Parallel Libraries

The new standard for parallel programming?

- Cilk, Intel TBB, Microsoft TPL, KAAPI, ...
- The programmer declares tasks and dependencies
- Tasks can be created at runtime
- Advantages:
  - easier to program (less error-prone than threads)
  - platform independent
  - the library is in charge of load balancing

Greedy tasks scheduler

- When tasks are available, no processor is idle.
- Graham's guarantee: $C_{\max} \leq \frac{W}{m} + (1 - \frac{1}{m}) \cdot T_\infty$

How to manage the tasks efficiently at runtime?

Global list of tasks

- Tasks generated by a running task are inserted in the list
- When a processor is idle, it retrieves a task from the list

Problem: concurrent accesses

- The list is accessed concurrently by several processors
- Protect the list by a lock or use a lock-free list
- Does not scale well in practice for small grain tasks

Efficient Tasks Management by Work Stealing

Decentralize the list

- Each processor has its own list
- If empty, it tries to **steal** tasks in others' lists
- The scheduler is no longer greedy:
  A processor can be idle while tasks are available in others' lists

Previous work

- Work generation is probabilist, focus on steady state results
  [Mitzenmacher 98, Berenbrink *et al.* 03]
- So far, only bound on $C_{\max}$ is in the Cilk model
  [Blumofe Leiserson 99, Arora *et al.* 01]
  - $\mathbb{E}[C_{\max}] \leq \frac{W}{m} + O(T_\infty)$
  - Unit tasks, DAG with only one source and out-degree at most 2

$\Longrightarrow$ Need for a precise analysis of work stealing for $P|prec, p_j|C_{\max}$

This Talk

Results

- Bound $C_{\max}$ of work stealing for $P|p_j|C_{\max}$
- Almost tight analysis (independent tasks and Cilk model)

Work In Progress

- We still don't have a tight analysis of WS for $P|prec, p_j|C_{\max}$

Overview

1. Model and notations
2. Sketch of the analysis
   1. Define a potential function
   2. Compute the expected decrease of the potential in one step
   3. Study a probabilistic game
3. Simulation results

Model and Notations

- Platform with m synchronized and identical processors
- Workload W of n independent tasks with processing times p_j
- Each processor owns a list of tasks
- An active processor (non-empty list) executes one unit of work
- An idle processor (thief) randomly chooses another processor (victim)
  - If the victim's list is non empty, the thief steals half of the tasks and resumes execution at the next time slot
  - Otherwise, the thief tries again at the next time slot
- Contention on lists:
  if several thieves target the same victim a random succeed, others fail.

Model and Notations

Amount of work in list Q_i of processor i at t $\quad w_i(t) = \sum_{j \in Q_i(t)} p_j$

Total amount of work at t $\quad w(t) = \sum_{1 \le i \le m} w_i(t)$

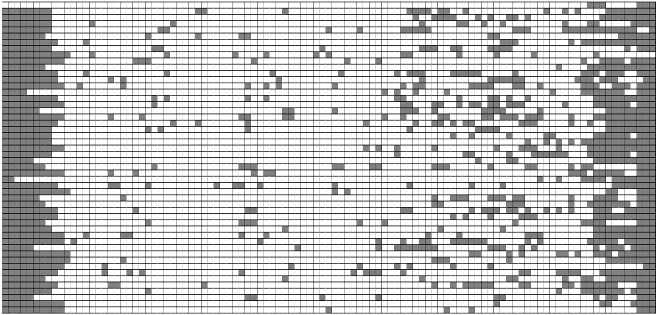
Total amount of work $\quad W = w(0)$

Number of active processors at t $\quad \alpha(t)$

Number of idle processors at t $\quad m - \alpha(t)$

Total number of steals $\quad S = \sum_{1 \le t \le C_{\max}} m - \alpha(t)$

Potential Function Φ: Motivation



Gantt chart with 25 processors and 2000 unit tasks
White: execution Grey: steal

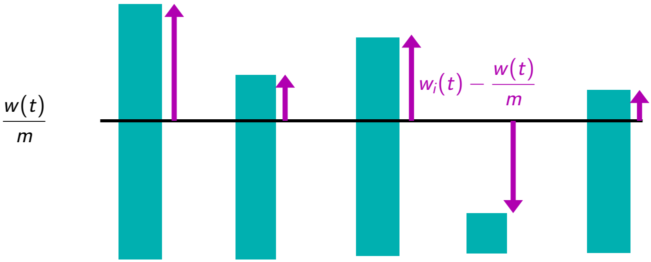
- $m \cdot C_{\max} = W + S$
- Bound S to bound $C_{\max}$
- Difficult to see any structure due to the random choices
- Potential function decreasing at each successful steal

Potential Function Φ: Definition

Definition

$$\Phi(t) = \sum_{1 \le i \le m} \left(w_i(t) - \frac{w(t)}{m} \right)^2$$

Φ represents how well the load is balanced between the lists



Potential Function Φ: Properties

$$\Phi(t) = \sum_{1 \leq i \leq m} \left(w_i(t) - \frac{w(t)}{m} \right)^2$$

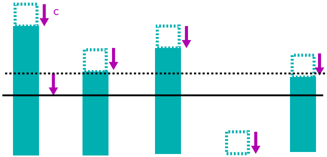
1. $\Phi = 0 \implies$ no more steals

Potential Function Φ: Properties

$$\Phi(t) = \sum_{1 \le i \le m} \left(w_i(t) - \frac{w(t)}{m} \right)^2$$

1. $\Phi = 0 \Longrightarrow$ no more steals

2. $\forall i, w_i \rightarrow w_i - c \Longrightarrow \Delta\Phi = 0$

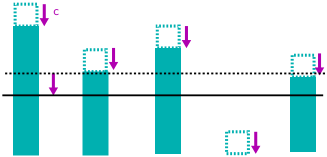


Potential Function Φ: Properties

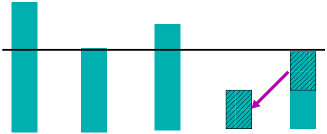
$$\Phi(t) = \sum_{1 \leq i \leq m} \left(w_i(t) - \frac{w(t)}{m} \right)^2$$

1. $\Phi = 0 \implies$ no more steals

2. $\forall i, w_i \to w_i - c \implies \Delta\Phi = 0$



3. Idle processor i steals half of the work of active processor $j \implies \Delta\Phi = \frac{w_j^2}{2}$



Expected decrease of Φ in one step

- Reminder: contention on the lists, only one steal succeed
- Decompose the potential decrease $\Delta\Phi$ per active processor

$$\Phi(t) = \sum_{1 \leq i \leq m} \left(w_i(t) - \frac{w(t)}{m} \right)^2 = \sum_{1 \leq i \leq m} w_i^2(t) - \frac{w^2(t)}{m}$$

$$\Delta\Phi(t) = \Phi(t) - \Phi(t+1) = \sum_{\text{active processors}} \delta_i(t) - \frac{1}{m} \cdot \Delta w^2(t)$$

- $\delta_i(t)$ is the decrease of $\sum w_i^2(t)$ on active processor i
- As $w(t+1) = w(t) - \alpha(t)$, we have
  $\Delta w^2(t) = w^2(t) - (w(t) - \alpha(t))^2 = 2\alpha(t)w(t) - \alpha^2(t)$

Expected decrease of Φ in one step

- If processor i is not stolen, one unit of work is executed

$$\begin{aligned}\delta_i(t) &= w_i^2(t) - w_i^2(t+1) \\ &= w_i^2(t) - (w_i(t)-1)^2 \\ &= 2w_i(t) - 1\end{aligned}$$

Expected decrease of Φ in one step

- If processor i is not stolen, one unit of work is executed

$$\begin{aligned}\delta_i(t) &= w_i^2(t) - w_i^2(t+1) \\ &= w_i^2(t) - (w_i(t)-1)^2 \\ &= 2w_i(t) - 1\end{aligned}$$

- If processor j steals half of the work of processor i

$$\begin{aligned}\delta_i(t) &= w_i^2(t) - w_i^2(t+1) - w_j^2(t+1) \\ &= w_i^2(t) - \left(\frac{w_i(t)}{2} - 1\right)^2 - \left(\frac{w_i(t)}{2}\right)^2 \\ &= \frac{w_i^2(t)}{2} + w_i(t) - 1\end{aligned}$$

Expected decrease of Φ in one step

- Expected decrease on active processor i

$$\mathbb{E}[\delta_i(t)] = \mathbb{P}\Big\{\text{processor } i \text{ is not stolen}\Big\} \cdot \Big(2w_i(t) - 1\Big)$$
$$+ \mathbb{P}\Big\{\text{processor } i \text{ is stolen}\Big\} \cdot \Big(\frac{w_i^2(t)}{2} + w_i(t) - 1\Big)$$

- As there are $m - \alpha(t)$ idle processors attempting to steal,

$$\mathbb{P}\Big\{\text{processor } i \text{ is stolen}\Big\} = p(\alpha(t)) = 1 - \Big(1 - \frac{1}{m-1}\Big)^{m-\alpha(t)}$$

- Summing δ_i on all active processors, we get

$$\mathbb{E}[\Delta\Phi(t)] \geq \frac{p(\alpha(t))}{2} \cdot \Phi(t)$$

Probabilistic Game

- We have the expected decrease of the potential in one step
  $\mathbb{E}[\Delta\Phi] \geq \frac{p(\alpha)}{2} \cdot \Phi$
- Problem: we don't know α

Probabilistic Game

- We have the expected decrease of the potential in one step $\mathbb{E}[\Delta\Phi] \geq \frac{p(\alpha)}{2} \cdot \Phi$
- Problem: we don't know α
- An adversary chooses the sequence $\alpha(t)$ maximizing the number of steals S
- At each time step t, the adversary chooses α, generating $m - \alpha$ steals but reducing the potential by $\Delta\Phi(\alpha)$

$$S \leftarrow S + m - \alpha$$
$$\Phi \leftarrow \Phi - \Delta\Phi(\alpha)$$

- The game ends when $\Phi \leq 1$

Probabilistic Game

- $S(\Phi)$: number of steals starting with a potential Φ to the end
- Dynamic Programming:

$$S(\Phi) = \max_{1 \leq \alpha \leq m-1} \left\{ m - \alpha + S(\Phi - \Delta\Phi(\alpha)) \right\}$$

- Problem: $\Delta\Phi(\alpha)$ is a random variable

Probabilistic Game

- $S(\Phi)$: number of steals starting with a potential Φ to the end
- Dynamic Programming:

$$S(\Phi) = \max_{1 \leq \alpha \leq m-1} \left\{ m - \alpha + S(\Phi - \Delta\Phi(\alpha)) \right\}$$

- Problem: $\Delta\Phi(\alpha)$ is a random variable
- Markov Decision Process:

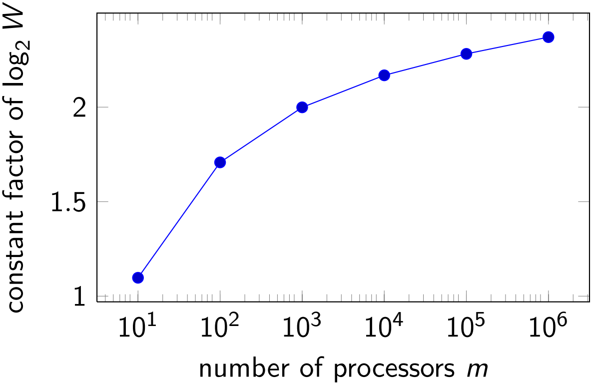
$$\mathbb{E}[S(\Phi)] = \max_{1 \leq \alpha \leq m-1} \left\{ m - \alpha + \mathbb{E}[S(\Phi - \Delta\Phi(\alpha))] \right\}$$

- Backwards induction:

$$\mathbb{E}[S] \leq \frac{m-1}{1 - \log_2(1 + \frac{1}{e})} \cdot \log_2 \Phi(0)$$

$$\mathbb{E}[C_{\max}] \leq \frac{W}{m} + 3.65 \cdot \log_2 W$$

Results from simulation



- Simulator strictly following the model
- Varying number of processors m, W unit tasks ($W = 8m$)
- 10000 experiments for each point
- Simulation: 2.37 vs. Our analysis: 3.65 (gap: adversary)

Summary of Results

Unit tasks

- Steal half of the tasks ($\approx$ half of the work due to rounding)
- $\mathbb{E}[C_{\max}] \leq \dfrac{W}{m} + 3.65 \cdot \log_2 W$
- Deviation from the mean

$$\mathbb{P}\Big\{ C_{\max} \geq \frac{W}{m} + 3.65 \cdot \Big(\log_2 W + \log_2 \frac{1}{\epsilon} \Big) \Big\} \leq \epsilon$$

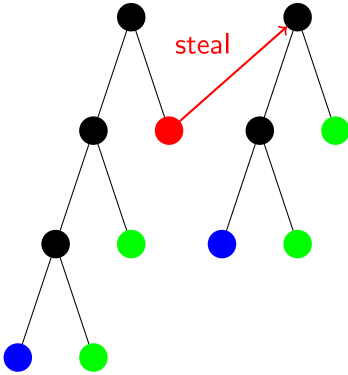
Weighted tasks

- Processing times are unknown
- Steal half of the tasks ($\neq$ half of the work)
- $\mathbb{E}[C_{\max}] \leq \dfrac{W}{m} + 4.1 \cdot \dfrac{p_{\max}}{p_{\min}} \cdot \log_2 W$

Summary of Results

Cilk model

- Unit tasks, online DAG, one source, out-degree at most 2
- Execute depth-first and steal breadth-first
- Previous analysis: based on critical path [Arora *et al.* 01]
    - $\mathbb{E}[C_{\max}] \leq \frac{W}{m} + 32 \cdot T_\infty$
    - $\mathbb{P}\left\{C_{\max} \geq \frac{W}{m} + 64 \cdot T_\infty + 16 \cdot \log_2 \frac{1}{\epsilon}\right\} \leq \epsilon$
- Our analysis: based on load balancing
    - $\mathbb{E}[C_{\max}] \leq \frac{W}{m} + 3.65 \cdot T_\infty$
    - $\mathbb{P}\left\{C_{\max} \geq \frac{W}{m} + 3.65 \cdot \left(T_\infty + \log_2 \frac{1}{\epsilon}\right)\right\} \leq \epsilon$



running task
ready task
stolen task
executed task

Conclusion

Decentralized list scheduling with work stealing

- Introduced a new technique based on a potential function
- Analyzed weighted tasks
- Improved the bound in the Cilk model
- Precise analysis: bound on number of steals is only 50% off

Future work

- Using this technique, we were also able to analyze modifications of the work stealing
- We believe we can extend this work to $P|prec, p_j|C_{\max}$