Estimating Parallel Performance,
A Skeleton-Based Approach

Oleg Lobachev and Rita Loogen

HLPP 2010
September 25, 2010
Parallel Performance Measures

- Amdahl’s law
- speedup = seq. time / parallel time
- efficiency
- serial fraction
- isoefﬁciency, scaled speedup, etc.

⇒ yet another one. Why?
algorithmic skeletons = parallel algorithm abstractions
in FP: higher-order functions
skeletons as algorithm classification
e.g., map-like, divide and conquer, iteration

⇒ use skeletons to designate types of parallel computation
The Essential Idea I

- different types of parallel programs run differently
- ⇒ classification using skeletons
The Essential Idea I

- different types of parallel programs run differently
  - $\Rightarrow$ classification using skeletons
- parallel programs do various things
  - $\Rightarrow$ computation part $+$ additional parallel overhead
- additional parallel overhead is harmful
- name it “parallel penalty”
The Essential Idea II

- $n =$ input size, $p =$ number of processors
- $T(n) =$ seq. runtime
- assumption: $T(n) =$ total amount of work
- $T(n, p) =$ parallel runtime
- traditional view:
  $$T(n, p) = \frac{T(n)}{\text{speedup}(n, p)}$$
- our approach: $\bar{A}(n, p) =$ parallel penalty
  $$T(n, p) = \frac{T(n)}{p} + \bar{A}(n, p)$$
The Essential Idea III

- sequential time $T(n)$ and parallel penalty $\bar{A}(n, p)$ are of different nature
  $\Rightarrow$ predict them separately!
- use statistical methods
sequential time $T(n)$ and parallel penalty $\bar{A}(n, p)$ are of different nature
⇒ predict them separately!

use statistical methods

be aware: $\bar{A}(n, p)$ has two dimensions:
input size $n$ and number of processors $p$
⇒ fix one, predict the other
"learning"

- measure $T(n)$ for several values of $n$
- measure $T(n, p)$ for several values of $n$ or $p$
  keeping the other parameter fixed
- compute

$$\bar{A}(n, p) = T(n, p) - \frac{T(n)}{p}$$

prediction

- predict $T(n)$ for non-measured values of $n$
- predict $\bar{A}(n, p)$ for non-measured values of $n$ or $p$
- compute $T(n, p)$ from above estimations
parMap creates a process for each function application (task)

we assume: same time needed for each list element

\[ T(n) = nT(1) \]

parMap: \[ T(n, p) = \frac{n}{p} T(1) + \bar{A}(n, p) \]
parMap creates a process for each function application (task)

we assume: same time needed for each list element

\[ T(n) = nT(1) \]

\[ \text{parMap: } T(n, p) = \frac{n}{p} T(1) + \bar{A}(n, p) \]
Parallel Map Skeletons II

- **farm**: static task distribution
  - divide task list into blocks before the computation

- same *assumption*: same task size for all tasks

\[
\text{farm: } T(n, p) = T(n/p) + \bar{A}(n, p)
\]
Parallel Map Skeletons II

- **farm**: static task distribution
  divide task list into blocks before the computation

- same *assumption*: same task size for all tasks

\[
farm: \quad T(n, p) = T(n/p) + \bar{A}(n, p)
\]
Parallel Iteration

\[
\begin{array}{cccc}
1 & 2 & \ldots & p \\
p + 1 & p + 2 & \ldots & 2p \\
\vdots & \vdots & \ddots & \vdots \\
(l - 1)p + 1 & (l - 1)p + 2 & \ldots & lp \\
\end{array}
\]

\{ l \text{ times} \}

- parallel do-while
- \( lp \) iterations, \( p \) speculative tasks, \( l \) “rounds”
- \( s(n) \) cost for a single iteration

\[
T(n) = lp \; s(n)
\]

\[
T(n, p, l) = l \; s(n) + \overline{A}(n, p)
\]
How do we proceed?

- measure $T(n)$
- measure $T(n, p)$
- compute $\bar{A}(n, p)$ from them

scale w.r.t. input size
- $\hat{n} =$ non-measured input size
- estimate $T(\hat{n})$
- estimate $\bar{A}(\hat{n}, p)$
- compute $T(\hat{n}, p)$ from them

scale w.r.t. number of processors
- $\hat{p} =$ non-measured number of PEs
- $T(n)$ is known!
- estimate $\bar{A}(n, \hat{p})$
- compute $T(n, \hat{p})$ from them
How do we proceed?

- measure $T(n)$
- measure $T(n, p)$
- compute $\bar{A}(n, p)$ from them

scale w.r.t. input size
- $\hat{n} =$ non-measured input size
- estimate $T(\hat{n})$
- estimate $\bar{A}(\hat{n}, p)$
- compute $T(\hat{n}, p)$ from them

scale w.r.t. number of processors
- $\hat{p} =$ non-measured number of PEs
- $T(n)$ is known!
- estimate $\bar{A}(n, \hat{p})$
- compute $T(n, \hat{p})$ from them
How do we proceed?

- measure $T(n)$
- measure $T(n, p)$
- compute $\bar{A}(n, p)$ from them

scale w.r.t. input size

- $\hat{n} =$ non-measured input size
- estimate $T(\hat{n})$
- estimate $\bar{A}(\hat{n}, p)$
- compute $T(\hat{n}, p)$ from them

scale w.r.t. number of processors

- $\hat{p} =$ non-measured number of PEs
- $T(n)$ is known!
- estimate $\bar{A}(n, \hat{p})$
- compute $T(n, \hat{p})$ from them
### Statistical methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>spline</strong></td>
<td>interpolation method</td>
</tr>
<tr>
<td></td>
<td>exact in specified points</td>
</tr>
<tr>
<td></td>
<td>stepwise polynomials</td>
</tr>
<tr>
<td></td>
<td>weighted extrapolation</td>
</tr>
<tr>
<td><strong>loess</strong></td>
<td>“local polynomial regression fitting”</td>
</tr>
<tr>
<td></td>
<td>inexact: regression fitting</td>
</tr>
<tr>
<td></td>
<td>stepwise polynomials</td>
</tr>
<tr>
<td></td>
<td>special feature for extrapolation</td>
</tr>
<tr>
<td><strong>lm</strong></td>
<td>linear model fitting</td>
</tr>
<tr>
<td></td>
<td>inexact: regression fitting</td>
</tr>
<tr>
<td></td>
<td>fits a single line</td>
</tr>
<tr>
<td><strong>lm(poly)</strong></td>
<td>linear model fitting with orthogonal polynomials</td>
</tr>
<tr>
<td></td>
<td>inexact: regression fitting</td>
</tr>
<tr>
<td></td>
<td>relaxed with polynomials</td>
</tr>
<tr>
<td><strong>mean</strong></td>
<td>take average of the best two methods</td>
</tr>
</tbody>
</table>

---

Oleg Lobachev and Rita Loogen

*Estimating Parallel Performance, A Skeleton-Based Approach*
A speedup analogy

absolute speedup vs. relative speedup

\[ \frac{T(n, p)}{T(n)} \quad \text{vs.} \quad \frac{T(n, p)}{T(n, 1)} \]

need sequential time to compute \( \overline{A}(n, p) \)

absolute reference point vs. relative reference point

\[ \overline{A}(n, p) = T(n, p) - \frac{T(n)}{p} \quad \text{vs.} \quad \overline{A}(n, p) = T(n, p) - \frac{T(n, 1)}{p} \]
**Gauß Elimination: Predicting Sequential Time**

- **Matrix computations**
- **a farm instance**
- **predict** $T(n)$ **for** $n = 120, 150$
- **best: lm(poly)**
- $-3.41\%$ rel. err **for** $n = 120$
Gauß Elimination: Predicting Parallel Penalty

- predict $\bar{A}(n, p)$ w.r.t. $n$
  - with fixed $p = 7$
  - for $n = 120, 150$
- best: mean
- $-0.73\%$ rel. err
  - for $n = 120$
Gauß Elimination: Estimation results

- want parallel runtime $T(\hat{n}, p)$ for $\hat{n} = 120$ and fixed $p = 7$
- combine estimations of $T(\hat{n})$ and $\bar{A}(\hat{n}, 7)$ w.r.t. $\hat{n}$
  $\Rightarrow T(\hat{n}, p)$ with an relative error $-1.69\%$
- estimate $T(n, \hat{p})$ for non-measured $\hat{p}$: also possible
  $\Rightarrow$ see the paper, relative error $-1.1\%$
Gauß Elimination: Estimation results

- want parallel runtime $T(\hat{n}, p)$ for $\hat{n} = 120$
  and fixed $p = 7$
- combine estimations of $T(\hat{n})$ and $\bar{A}(\hat{n}, 7)$ w.r.t. $\hat{n}$
  $\Rightarrow T(\hat{n}, p)$ with an relative error $-1.69\%$
- estimate $T(n, \hat{p})$ for non-measured $\hat{p}$: also possible
  $\Rightarrow$ see the paper, relative error $-1.1\%$
- considered implementation of Gauß Elimination
  always spawns 8 tasks
- our method does not know this!
Serial Fraction

- **serial fraction** = measure for sequential part of the program
  \[ f(n, p) = \frac{T(n, p)}{T(n) - 1/p} \frac{1}{1 - 1/p} \]
- parallel program quality measure
- should be constant
Parallel penalty vs. serial fraction for Gauß Elimination

\[ A(n, p) \]

\[ f(n, p) \]

PEs

Oleg Lobachev and Rita Loogen

Estimating Parallel Performance, A Skeleton-Based Approach
Rabin-Miller Test

- Rabin-Miller test = instance of iteration skeleton
- checks for primality
- definite answer in negative case
- is never sure in positive case
- our implementation: speculative iteration
  - has load-balancing issues
  - always starts 20 tasks
- predict for an input size $n = 11213$
Results for Rabin-Miller Test

- Predicting $T(n)$
  - Best: `lm(poly)`

- Predicting $\bar{A}(n, p)$
  - Best: `mean`

Combined: 0.01% rel. err
Parallel penalty vs. serial fraction for Rabin-Miller test

- **Parallel penalty**

- **Serial fraction**

**A(n,p)**

**f(n,p)**

PEs
Conclusions and Future Work

Conclusions

- a method for parallel runtime estimation, better than a direct prediction for vast class of programs (more examples and experiments in the paper!)
- parallel penalty term as quality measure

Future work

- investigate relation to serial fraction further
- extend the formalism to further skeletons
- try other statistical methods
- try other prediction techniques — automated learning
Karatsuba multiplication: outline

- fast integer multiplication
- instance of divide and conquer skeleton
- used with a fine-granular divide and conquer skeleton
- implemented in Eden — parallel Haskell extension
- distributed memory setting
- tested on a multicore
<table>
<thead>
<tr>
<th>$n \cdot 1000$</th>
<th>16</th>
<th>$\ldots$</th>
<th>56</th>
<th>60</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n, 8)$</td>
<td>1.29</td>
<td>$\ldots$</td>
<td>9.95</td>
<td>11.0</td>
<td>11.86</td>
</tr>
<tr>
<td>$T(n)$</td>
<td>9.88</td>
<td>$\ldots$</td>
<td>74.39</td>
<td>82.02</td>
<td>88.94</td>
</tr>
<tr>
<td>$\bar{A}(n, 8) \cdot 100$</td>
<td>5.39</td>
<td>$\ldots$</td>
<td>64.66</td>
<td>74.22</td>
<td>74.65</td>
</tr>
<tr>
<td>predict $T(n)$, rel. err, %</td>
<td>$-0.014$</td>
<td>$\ldots$</td>
<td>$1.9$</td>
<td>2.3</td>
<td>2.08</td>
</tr>
<tr>
<td>predict $\bar{A}$, w.r.t. $n$, rel. err, %</td>
<td>$2.3$</td>
<td>$\ldots$</td>
<td>$2.08$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>predict $T(n, p)$, rel. err, %</td>
<td>$0.14$</td>
<td>$\ldots$</td>
<td>$1.78$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Karatsuba multiplication: estimating $T(n)$

![Graph showing estimated time versus input size](image)

- **measured time**
- **spline**
- **loess**
- **lm**
- **lm(poly)**

Input size, $n$

Time, seconds