Performance models for
master/slave parallel programs

PASM 2004

Lucas Baldo    Leonardo Brenner    Luiz G. Fernandes    Paulo Fernandes    Afonso Sales

{lbaldo, lbrenner, gustavo, paulof, asales}@inf.pucrs.br.

PUCRS - FACIN - PPGCC
Porto Alegre, 90619-900, Brazil
HP Brazil - CNPq/Brazil
Motivation

- Foresee implementation behavior in early stages of the parallel program development
  - Identification of performance indices
- Provide an easy-to-use (or at least feasible) stochastic modeling
  - Any modeling technique could be used (PEPA, SPN, etc)
  - Very dependable on programmer previous experience
Outline

- Stochastic Automata Networks (SAN)
- Case Study: Propagation Algorithm
- Proposed models
  - Asynchronous
  - Synchronous
- Performance Indices
- Conclusions
SAN (Concept)

- Proposed by B. Plateau
- Each subsystem is represented by a stochastic automaton
- Local events change the local state of only one automaton
- Synchronizing events change the local state of several automata simultaneously
- Interaction among automata:
  - Synchronizing events
  - Functional rates and/or functional probabilities
SAN (Example)

\[ f = \left[ (st \, A^{(2)} = 0^{(2)}) \ast \lambda \right] + \left[ (st \, A^{(2)} = 2^{(2)}) \ast \gamma \right] \]

Reachability = \(! \left[ (st \, A^{(1)} = 1^{(1)}) \&\& (st \, A^{(2)} = 1^{(2)}) \right] \)
Master/Slave Paradigm

- One master node generates and allocates work \((tasks)\) to \(N\) slave nodes.
- Each slave node receives tasks, computes them, and sends the result back.
- Master node receives and analyses the computed results by slaves \((summation)\).
Master/Slave Generic SAN Model

\begin{itemize}
  \item **Master**
  \begin{itemize}
    \item \textbf{up} begins the program execution
    \item \textbf{down} ends the parallel application (returns to the initial situation)
    \item \textbf{s} sends tasks to the slaves
    \item \textbf{c} performs the summation
    \item \textbf{p}_i finishes a task by the \textit{i}^{th} slave
    \item \textbf{r}_i sends the result from \textit{i}^{th} slave back to the master
  \end{itemize}
\end{itemize}

\begin{itemize}
  \item **Slave\textsuperscript{(i)}**
  \begin{itemize}
    \item \textbf{I} \textbf{Tx} \textbf{r}_1 \ldots \textbf{r}_N \textbf{Rx} \textbf{ITx}
    \end{itemize}
\end{itemize}
Case Study

- Image interpolation application
  - method to create smooth and realistic virtual views
  - starts with two source images

- Propagation Algorithm
  - input: seed pairs
  - based on a region growing technique
  - goal: match the largest possible region
Asynchronous Implementation

- One master, one buffer and $N$ slaves
- The results sent by the slaves are stored on a repository (**buffer**)
- Master does not need to synchronize the reception of results
SAN Model of Asynchronous Implementation

\[ g_1 = (nb [\text{Slave}^{(1)}..\text{Slave}^{(N)}] I == 0) \]

\[ g_2 = (nb [\text{Slave}^{(1)}..\text{Slave}^{(N)}] I > 0) \]

<table>
<thead>
<tr>
<th>Type</th>
<th>Event</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn</td>
<td>up</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>syn</td>
<td>( s_i )</td>
<td>( \delta )</td>
</tr>
<tr>
<td>syn</td>
<td>( r_i )</td>
<td>( \alpha )</td>
</tr>
<tr>
<td>syn</td>
<td>( p_i )</td>
<td>( \gamma )</td>
</tr>
<tr>
<td>syn</td>
<td>down</td>
<td>( \mu )</td>
</tr>
<tr>
<td>syn</td>
<td>loc</td>
<td>( \eta )</td>
</tr>
</tbody>
</table>

Type Event | Rate |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>syn up ( \lambda )</td>
<td>syn down ( \mu )</td>
</tr>
<tr>
<td>syn ( s_i ) ( \delta )</td>
<td>syn ( r_i ) ( \alpha )</td>
</tr>
</tbody>
</table>
Parameterization for Asynchronous Implementation

- Input values:

  \[ BL \] slaves buffer length (5.500 points)
  \[ PS \] percentage of slices extension over its neighborhoods (0.5)
  \[ NP \] number of points (230.000)
  \[ NS \] number of slices (granularity)
  \[ RNP \] real number of points \( \Rightarrow \left[ 2 \times (1 + PS) + (NS - 2) \times (1 + 2PS) \right] \times \frac{NP}{NS} \)
  \[ NB \] number of buffer for each slice \( \Rightarrow \left[ \frac{RNP}{BL \times NS} \right] \)
Parameterization for Asynchronous Implementation

Rates:

\[ s_i = \frac{1}{TT} \quad TT \Rightarrow \text{send a new task} \]
\[ \lambda = \frac{1}{TT \times N} \quad N \Rightarrow \text{number of slaves} \]
\[ \mu = 1000 \quad \text{(insignificant time)} \]
\[ \gamma = \frac{1}{CM} \quad CM \Rightarrow \text{compute final matches} \]
\[ \sigma = \frac{1}{ER} \quad ER \Rightarrow \text{evaluate results} \]
\[ \alpha = \frac{1}{TP} \quad TP \Rightarrow \text{send a result pack} \]
Synchronous Implementation

- One master and $N$ slaves
- The results are sending to the master
- Master waits all slaves have finished their tasks before it starts a new task distribution
SAN Model of Synchronous Implementation

\[ f_c = (nb [\text{Slave}^{(1)}..\text{Slave}^{(N)}]) \cdot \tau \]

<table>
<thead>
<tr>
<th>Type</th>
<th>Event</th>
<th>Rate</th>
<th>Type</th>
<th>Event</th>
<th>Rate</th>
<th>Type</th>
<th>Event</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>syn</td>
<td>up</td>
<td>(\lambda)</td>
<td>syn</td>
<td>down</td>
<td>(\mu)</td>
<td>loc</td>
<td>c</td>
<td>(f_c)</td>
</tr>
<tr>
<td>syn</td>
<td>(s)</td>
<td>(\delta)</td>
<td>syn</td>
<td>(r_i)</td>
<td>(\alpha)</td>
<td>loc</td>
<td>(p_i)</td>
<td>(\gamma)</td>
</tr>
</tbody>
</table>
Parameterization for Synchronous Implementation

Rates:

\[ s \delta = \frac{1}{TT} \quad TT \Rightarrow \text{send a new task} \]

\[ up \lambda = \frac{1}{TT \times N} \quad N \Rightarrow \text{number of slaves} \]

\[ down \mu = 1000 \quad \text{(insignificant time)} \]

\[ p_i \gamma = \frac{1}{CM} \quad CM \Rightarrow \text{compute final matches} \]

\[ c \sigma = \frac{1}{ER} \quad ER \Rightarrow \text{evaluate results} \]

\[ r_i \alpha = \frac{1}{TP \times N} \quad TP \Rightarrow \text{send a result pack} \]
Performance Indices

- Stationary analysis
- Local state probabilities:
  - Slave in $T_x$ state
    - master unavailable
  - Master in $R_x$ state
    - master unavailable
  - Master in $T_x$ state
    - slaves unavailable
Synchronous vs. Asynchronous

**Synchronous Model**

<table>
<thead>
<tr>
<th>$N$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>0.1746</td>
<td>0.2042</td>
<td>0.2225</td>
<td>0.2416</td>
<td>0.2533</td>
<td>0.2609</td>
</tr>
<tr>
<td>Fine</td>
<td>0.1687</td>
<td>0.1954</td>
<td>0.2125</td>
<td>0.2275</td>
<td>0.2362</td>
<td>0.2503</td>
</tr>
</tbody>
</table>

**Asynchronous Model**

<table>
<thead>
<tr>
<th>$N$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>0.1967</td>
<td>0.1604</td>
<td>0.1417</td>
<td>0.1395</td>
<td>0.1725</td>
<td>0.2300</td>
</tr>
<tr>
<td>Fine</td>
<td>0.1769</td>
<td>0.1499</td>
<td>0.1355</td>
<td>0.1336</td>
<td>0.1651</td>
<td>0.2253</td>
</tr>
</tbody>
</table>
Buffer Size

Asynchronous Implementation with coarse grain

<table>
<thead>
<tr>
<th>Buffer size</th>
<th>2 Slaves</th>
<th>3 Slaves</th>
<th>4 Slaves</th>
<th>5 Slaves</th>
<th>6 Slaves</th>
<th>7 Slaves</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3696</td>
<td>0.3729</td>
<td>0.3940</td>
<td>0.4264</td>
<td>0.4643</td>
<td>0.5025</td>
</tr>
<tr>
<td>5</td>
<td>0.2935</td>
<td>0.2443</td>
<td>0.2404</td>
<td>0.2799</td>
<td>0.3423</td>
<td>0.4061</td>
</tr>
<tr>
<td>10</td>
<td>0.2763</td>
<td>0.2190</td>
<td>0.1956</td>
<td>0.2266</td>
<td>0.2963</td>
<td>0.3680</td>
</tr>
<tr>
<td>20</td>
<td>0.2477</td>
<td>0.1953</td>
<td>0.1668</td>
<td>0.1765</td>
<td>0.2380</td>
<td>0.3103</td>
</tr>
<tr>
<td>40</td>
<td>0.2099</td>
<td>0.1696</td>
<td>0.1492</td>
<td>0.1434</td>
<td>0.1761</td>
<td>0.2341</td>
</tr>
</tbody>
</table>

Asynchronous Implementation with fine grain

<table>
<thead>
<tr>
<th>Buffer size</th>
<th>2 Slaves</th>
<th>3 Slaves</th>
<th>4 Slaves</th>
<th>5 Slaves</th>
<th>6 Slaves</th>
<th>7 Slaves</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3630</td>
<td>0.3652</td>
<td>0.3846</td>
<td>0.4176</td>
<td>0.4542</td>
<td>0.4954</td>
</tr>
<tr>
<td>5</td>
<td>0.2760</td>
<td>0.2280</td>
<td>0.2237</td>
<td>0.2672</td>
<td>0.3308</td>
<td>0.3999</td>
</tr>
<tr>
<td>10</td>
<td>0.2493</td>
<td>0.1969</td>
<td>0.1754</td>
<td>0.2116</td>
<td>0.2830</td>
<td>0.3703</td>
</tr>
<tr>
<td>20</td>
<td>0.2129</td>
<td>0.1708</td>
<td>0.1478</td>
<td>0.1626</td>
<td>0.2249</td>
<td>0.3017</td>
</tr>
<tr>
<td>40</td>
<td>0.1769</td>
<td>0.1499</td>
<td>0.1355</td>
<td>0.1336</td>
<td>0.1651</td>
<td>0.2253</td>
</tr>
</tbody>
</table>

Asynchronous (Coarse)

Asynchronous (Fine)
Granularity

Asynchronous Model

<table>
<thead>
<tr>
<th>( N )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>0.9030</td>
<td>0.9320</td>
<td>0.9350</td>
<td>0.9295</td>
<td>0.9190</td>
<td>0.9074</td>
</tr>
<tr>
<td>Fine</td>
<td>0.9202</td>
<td>0.9348</td>
<td>0.9282</td>
<td>0.9152</td>
<td>0.8970</td>
<td>0.8940</td>
</tr>
</tbody>
</table>

Receiving

<table>
<thead>
<tr>
<th>( N )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>0.0887</td>
<td>0.0595</td>
<td>0.0564</td>
<td>0.0620</td>
<td>0.0725</td>
<td>0.0843</td>
</tr>
<tr>
<td>Fine</td>
<td>0.0713</td>
<td>0.0566</td>
<td>0.0633</td>
<td>0.0764</td>
<td>0.0947</td>
<td>0.0978</td>
</tr>
</tbody>
</table>

Transmitting
Conclusions

We can:

- Choose the paradigm
- Choose the buffer size
- Choose the granularity
- Find possible bottlenecks before implementation

We cannot (future work):

- Estimate execution time
  - Transient analysis
- Provide an easy-to-use stochastic modeling
  - Automatic modeling tool
Contact

Performance Evaluation Group

www.inf.pucrs.br/peg