Cutting Out the Middleman: OS-Level Support for X10 Activities

Manuel Mohr, Sebastian Buchwald, Andreas Zwinkau, Christoph Erhardt, Benjamin Oechslein, Jens Schedel, Daniel Lohmann
Ideal World

X10

OS

HW
Ideal World

X10

OS

HW
Ideal World
Current Practice

X10

OS

HW
Current Practice

X10

OS

HW
Current Practice

X10

RT

OS

HW
Current Practice

```
sleep(100)
```

<table>
<thead>
<tr>
<th>X10</th>
<th>RT</th>
<th>OS</th>
<th>HW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(KIT)
Current Practice

```
sleep(100)
```

<table>
<thead>
<tr>
<th>X10</th>
<th>RT</th>
<th>OS</th>
<th>HW</th>
</tr>
</thead>
</table>

**IPD, CS4**

**KIT**

---

Manuel Mohr, Sebastian Buchwald, Andreas Zwinkau, Christoph Erhardt, Benjamin Oechslein, Jens Schedel, Daniel Lohmann – Cutting Out the Middleman: OS-Level Support for X10 Activities
class Runtime {
    public static def sleep(millis:Long) {
        Runtime.increaseParallelism();
        Thread.sleep(millis);
        Runtime.decreaseParallelism(1);
    }
}
Workaround in Action
Workaround in Action

\[ \text{sleep}(100) \]

X10 \rightarrow RT \rightarrow OS \rightarrow HW
Workaround in Action

sleep(100)

X10

RT

OS

HW
Workaround in Action

sleep(100)

X10

RT

OS

HW
Motivation

Problems with user-level scheduling approach:
- Complexity: interplay between two schedulers
- Performance: starting/stopping kernel-level threads is expensive
- Bugs: what if starting/stopping is forgotten? (e.g., user code)
Motivation

Problems with user-level scheduling approach:
- Complexity: interplay between two schedulers
- Performance: starting/stopping kernel-level threads is expensive
- Bugs: what if starting/stopping is forgotten? (e.g., user code)

⇒ Why not activity = OS-level primitive?
Motivation

Problems with user-level scheduling approach:
- Complexity: interplay between two schedulers
- Performance: starting/stopping kernel-level threads is expensive
- Bugs: what if starting/stopping is forgotten? (e.g., user code)

⇒ Why not activity = OS-level primitive?

In this talk:
- How we directly mapped activities to OS primitives
  - Context: many-core hardware architecture
- How this simplifies runtime system and OS
- Initial evaluation of system efficiency
Tiled Many-Core Architectures
OS designed for many-core PGAS architectures
OS designed for many-core PGAS architectures

**PGAS Architecture**

⇒ One OS instance per place
⇒ Message passing
OS designed for many-core PGAS architectures

Many-Core
⇒ Enough cores for exclusive allocation
⇒ Cooperative scheduling instead of preemption

PGAS Architecture
⇒ One OS instance per place
⇒ Message passing
OS designed for many-core PGAS architectures

Many-Core

⇒ Enough cores for exclusive allocation

⇒ Cooperative scheduling instead of preemption

PGAS Architecture

⇒ One OS instance per place

⇒ Message passing

User-level-like scheduler in the kernel

⇒ Cooperative FIFO scheduling
OS designed for many-core PGAS architectures

Many-Core

⇒ Enough cores for *exclusive* allocation
⇒ Cooperative scheduling instead of preemption

PGAS Architecture

⇒ One OS instance per place
⇒ Message passing

User-level-like scheduler in the kernel

⇒ Cooperative FIFO scheduling
⇒ Very lightweight threads called *i-lets*
Async

X10

RT

OS

HW

Very thin runtime system, no user-level scheduler
Blocking calls unproblematic, no workaround needed
Each activity corresponds to exactly one i-let. Very thin runtime system, no user-level scheduler. Blocking calls unproblematic, no workaround needed.
Each activity corresponds to exactly one i-let

- Very thin runtime system, no user-level scheduler
- Blocking calls unproblematic, no workaround needed
Remote \textit{i}-let spawning

\texttt{spawn\_ilet(place\_id, ilet)}

- Start an \textit{i}-let on a different place
- Asynchronous
Small At Async Statement: at (B) async S

i-let

B
Small At Async Statement: \( \text{at (B) async S} \)
Small At Async Statement: at (B) async $S$

$i$-let

$B$

exec $S$
Small At Async Statement: at (B) async S
Push DMA Transfer

push_dma(place_id, data, length, sender_iilet, receiver_iilet)
- Copy memory block to different place
- Specify actions to be executed when transfer is finished
- Asynchronous, HW support
At Async Statement: \texttt{at (B) async S}

\begin{itemize}
  \item \texttt{i-let}
  \item \texttt{A}
  \item \texttt{DMA}
  \item \texttt{B}
\end{itemize}
At Async Statement: at (B) async S
At Async Statement: \texttt{at (B) async S}

\begin{itemize}
\item \texttt{i-let}
\item \texttt{A}
\item \texttt{DMA}
\item \texttt{B}
\end{itemize}
At Async Statement: at (B) async S

```
i-let
A
DMA
B
```
At Async Statement: at (B) async S

\[
i\text{-let} \quad A \quad \text{DMA} \quad B
\]

exec S
At Async Statement: \texttt{at (B) async S}

\begin{itemize}
  \item \texttt{i-let}
  \item \texttt{A}
  \item \texttt{DMA}
  \item \texttt{B}
\end{itemize}

\texttt{exec S}
At Async Statement: \texttt{at (B) async S}

\begin{itemize}
\item \texttt{i-let}
\item \texttt{A}
\item \texttt{DMA}
\item \texttt{B}
\end{itemize}

\texttt{exec S}

\texttt{global termination}
Evaluation

- Benchmarks on FPGA-based prototype hardware
  - 4 places with 4 cores each
  - 25 MHz
Evaluation

- Benchmarks on FPGA-based prototype hardware
  - 4 places with 4 cores each
  - 25 MHz
- Measurements (in clock cycles):

<table>
<thead>
<tr>
<th>Function Description</th>
<th>Measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spawn_ilet(0, ilet)</code></td>
<td>539</td>
</tr>
<tr>
<td><code>async {}</code></td>
<td></td>
</tr>
<tr>
<td><code>spawn_ilet(1, ilet)</code></td>
<td>1133</td>
</tr>
<tr>
<td><code>at (Place(1)) async {}</code></td>
<td>1981</td>
</tr>
</tbody>
</table>

Operations are cheap (in absolute numbers)
**Evaluation**

- Benchmarks on FPGA-based prototype hardware
  - 4 places with 4 cores each
  - 25 MHz

- Measurements (in clock cycles):

<table>
<thead>
<tr>
<th>Spawn_ilet(0, it) async {}</th>
<th>539</th>
<th>1469</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spawn_ilet(1, it) at (Place(1)) async {}</td>
<td>1133</td>
<td>1981</td>
</tr>
</tbody>
</table>

Operations are cheap (in absolute numbers)
Evaluation

- Benchmarks on FPGA-based prototype hardware
  - 4 places with 4 cores each
  - 25 MHz

- Measurements (in clock cycles):

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>spawn_ilet(0, ilet)</td>
<td>async { }</td>
</tr>
<tr>
<td>539</td>
<td>1469</td>
</tr>
<tr>
<td>spawn_ilet(1, ilet)</td>
<td>at (Place(1))</td>
</tr>
<tr>
<td>1133</td>
<td>async { }</td>
</tr>
<tr>
<td></td>
<td>1981</td>
</tr>
</tbody>
</table>

- Operations are cheap (in absolute numbers)
Conclusion & Future Work

We have:
- Implemented X10 activity management without a user-level scheduler
  - Possible by exclusively allocating cores and using cooperative scheduling
  - Essentially puts user-level-like scheduler into kernel
- Adapted the X10 runtime
- Evaluated the efficiency on a prototype many-core architecture

We plan to:
- Port OctoPOS to AMD64 NUMA systems (in progress)
- Evaluate against common Linux-MPI implementations
Conclusion & Future Work

We have:
- Implemented X10 activity management without a user-level scheduler
  - Possible by exclusively allocating cores and using cooperative scheduling
  - Essentially puts user-level-like scheduler into kernel
- Adapted the X10 runtime
- Evaluated the efficiency on a prototype many-core architecture

We plan to:
- Port OctoPOS to AMD64 NUMA systems (in progress)
- Evaluate against common Linux-MPI implementations
Backup Slides
At Expression: at (B) E
At Expression: \textit{at (B) E}
At Expression: at (B) E

- i-let
- A
- DMA
- B

blocks until loc. term.
At Expression: at (B) E

- i-let
- A
- DMA
- B

blocks until loc. term.
At Expression: \textit{at (B) E}

\begin{itemize}
  \item \textit{i-let}
  \item A
  \item DMA
  \item B
\end{itemize}

blocks until loc. term.
At Expression: \textbf{at (B) E}

\begin{itemize}
  \item \texttt{i-let}
  \item A
  \item DMA
  \item B
\end{itemize}

- \textit{blocks until loc. term.}
- \textit{exec at body}
At Expression: \texttt{at (B) E}

\begin{center}
\begin{tikzpicture}
  \node (i-let) at (0,0) {i-let};
  \node (A) at (1,0) {A};
  \node (DMA) at (2,0) {DMA};
  \node (B) at (3,0) {B};

  \draw[dotted] (i-let) -- (A);
  \draw[dotted] (A) -- (DMA);
  \draw[dotted] (DMA) -- (B);

  \draw[fill=black!20] (i-let) rectangle (0.2,0);
  \draw[fill=black!20] (A) rectangle (1.2,0);
  \draw[fill=black!20] (DMA) rectangle (2.2,0);
  \draw[fill=black!20] (B) rectangle (3.2,0);

  \node at (0.4,-0.4) {blocks until loc. term.};
  \node at (1.8,-0.4) {exec at body};
\end{tikzpicture}
\end{center}
At Expression: \texttt{at (B) E}

- \texttt{i-let}
- \texttt{A}
- \texttt{DMA}
- \texttt{B}

blocks until loc. term.

exec at body
At Expression: \texttt{at (B) E}

- \texttt{i-let}
- \texttt{A}
- \texttt{DMA}
- \texttt{B}

blocks until loc. term.

exec at body
At Expression: \textit{at (B) E}

\begin{itemize}
  \item \textit{i-let}
  \item A
  \item DMA
  \item B
\end{itemize}

Diagram:

- Blocks until loc. term.
- Exec at body
At Expression: \texttt{at (B) E}
At Expression: \texttt{at \ (B) E}

\begin{center}
\begin{tikzpicture}[node distance=1.5cm,>=latex]
  \node [block] (i-let) {\texttt{i-let}};
  \node [block, right of=i-let] (A) {A};
  \node [block, right of=A] (DMA) {DMA};
  \node [block, right of=DMA] (B) {B};

  \draw [->] (i-let) -- (A) node [midway, above] {blocks until loc. term.};
  \draw [->] (A) -- (DMA) node [midway, above] {exec at body};
  \draw [->] (DMA) -- (B) node [midway, above] {loc. term.};
\end{tikzpicture}
\end{center}
At Expression: \texttt{at (B) E}

\begin{itemize}
  \item \texttt{i-let}
  \item \texttt{A}
  \item \texttt{DMA}
  \item \texttt{B}
\end{itemize}