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

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

Chair for Programming Paradigms, Karlsruhe Institute of Technology (KIT) & System Software Group, University Erlangen-Nuremberg (FAU)
Ideal World

X10

OS

HW
Ideal World

X10

OS

HW
Ideal World
Current Practice

X10

OS

HW
Current Practice

X10

RT

OS

HW

[Diagram showing a hierarchy between X10, RT, OS, and HW with arrows indicating relationships or flows between them.]
Current Practice

```
sleep(100)
```

![Diagram](image)
Current Practice

```python
sleep(100)
```
class Runtime {
    public static def sleep(millis:Long) {
        Runtime.increaseParallelism();
        Thread.sleep(millis);
        Runtime.decreaseParallelism(1);
    }
}
Workaround in Action

X10

RT

OS

HW
Workaround in Action

```
sleep(100)
```

X10

RT

OS

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

[Diagram showing a tiled many-core architecture with cores, memory, and cache connections marked as 00, 01, 10, and 11.]
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
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
Each activity corresponds to exactly one i-let

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

\[ \text{spawn\_ilet(place\_id, ilet)} \]
- Start an i-let on a different place
- Asynchronous
Small At Async Statement: \texttt{at (B) async S}
Small At Async Statement: \( \text{at (B) async S} \)
Small At Async Statement: \( \text{at (B) async } S \)
Small At Async Statement: \texttt{at (B) async S}

\begin{center}
\begin{tikzpicture}
  \node[rectangle, draw] (i) {\texttt{i-let}};
  \node[rectangle, draw, right of=i, xshift=2cm] (B) {\texttt{B}};
  \node[rectangle, draw, below of=i, yshift=-2cm] (exec) {\texttt{exec S}};
  \node[rectangle, draw, below of=B, yshift=-2cm] (termination) {\texttt{global termination}};
  \draw[->] (i) -- (exec);
  \draw[->] (exec) -- (termination);
\end{tikzpicture}
\end{center}
Push DMA Transfer

push_dma(place_id, data, length, sender_ilet, receiver_ilet)

- Copy memory block to different place
- Specify actions to be executed when transfer is finished
- Asynchronous, HW support
At Async Statement: at (B) async S

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

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

\begin{center}
\begin{tikzpicture}
    \node[rectangle, draw, minimum width=2cm] (i-let) at (0,0) {\texttt{i-let}};
    \node[rectangle, draw, minimum width=2cm] (A) at (3,0) {A};
    \node[rectangle, draw, minimum width=2cm] (DMA) at (6,0) {DMA};
    \node[rectangle, draw, minimum width=2cm] (B) at (9,0) {B};

    \draw[->] (i-let) -- (A);
    \draw[->] (A) -- (DMA);
    \draw[->] (DMA) -- (B);
\end{tikzpicture}
\end{center}
At Async Statement: \texttt{at \{B\} \textbf{async S}}
At Async Statement: at (B) async S

i-let  A  DMA  B

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

\begin{center}
\begin{tabular}{cccc}
\textit{i}-let & A & DMA & B \\
\end{tabular}
\end{center}

\begin{center}
\begin{tikzpicture}
\node (i) at (0,0) {i-let};
\node (a) at (1,0) {A};
\node (d) at (2,0) {DMA};
\node (b) at (3,0) {B};
\node (s) at (4,0) {exec S};
\draw[->] (i) -- (a);
\draw[->] (a) -- (d);
\draw[->] (d) -- (b);
\draw[->] (b) -- (s);
\end{tikzpicture}
\end{center}
At Async Statement: \texttt{at (B) async S}
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>Method</th>
<th>Measurements (in clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spawn_ilet(0, ilet)</code> async {}</td>
<td>539</td>
</tr>
<tr>
<td><code>spawn_ilet(1, ilet)</code> at (Place(1)) async {}</td>
<td>1133</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, ilet) async {}</th>
<th>1469</th>
</tr>
</thead>
<tbody>
<tr>
<td>spawn_ilet(1, ilet) at (Place(1)) async {}</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>Function</th>
<th>Async Code</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>spawn_ilet(0, ilet) async {}</td>
<td></td>
<td>539</td>
</tr>
<tr>
<td>spawn_ilet(1, ilet) async {}</td>
<td>at (Place(1))</td>
<td>1133</td>
</tr>
</tbody>
</table>

- Operations are cheap (in absolute numbers)
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
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: \textit{at (B) E}
At Expression: \(\text{at (B) E}\)
At Expression: at (B) E
At Expression: \texttt{at (B) E}

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

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

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

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

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

blocks until loc. term.

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

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

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

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

\begin{itemize}
\item blocks until loc. term.
\item exec at body
\end{itemize}
At Expression: at (B) E
At Expression: \textbf{at (B) E}

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

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

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

- blocks until loc. term.
- loc. term.
- glob. term.
- exec at body