Invasive Computing: A Systems-Programming Perspective

Wolfgang Schröder-Preikschat
Collaborative Research Centre/Transregio

12 chairs
+60 scientists
22.5 funded thereof
term: 3 x 4 years
phase 1: €9m
started Q3/2010

Embedded Systems
Theoretical Informatics
Programming Paradigms
Information Processing Technologies
Humanoids and Intelligent Systems

* Distributed Systems & Operating Systems
  Hardware-Software-Codesign

* System software:
  5 scientists (full-time equiv.)
  1 system programmer
  4 funded thereof

Scientific Computing
Integrated Systems
Technical Electronics
Electronic Design Automation
Computer Engineering & Organization
Parallel Processing
Parallel Systems

Tile GX 100
source: Tilera
Parallel Systems (cont.)
Parallel Systems (cont.)
Levels of Parallelism

- **Process-level, thread-level**: Hardware + Software Control
  - Multi-Core

- **Loop-level**: For i=0 to N do
  - For j=0 to M do
  - ...

- **Instruction-level**: ADD R1, R2, R3
  - MUL R4, R1, S4
  - JMP $42

- **Word-level, bit-level**: 01010001 1010101010101010
  - 10101010100011111111

Granularity of parallelism vs. extent of parallelism
Invasive Computing
Acronym: InvasIC

Investigation of a new paradigm of parallel computing

• through introduction of resource-aware language, programming and operating-system support

• and through dynamic and distributed allocation and reconfiguration of processor, interconnect, and memory resources

• with particular focus on Multiprocessor System-on-Chip (SoC) systems for the years 2020 and beyond.
Resource-Aware

(Trinity of requisition, usage, and restoration)

1. invade
   - request/reserve computing resources

2. infect
   - deal with the resources received

3. retreat
   - release the resources received
Units of Invasion

Program section being aware of potential parallel execution

1. candidate
2. instance
3. incarnation
4. execution

"invasive-let" "i-let"

operating-system entities
System Abstractions

1. claim
   - of (hardware/software) resources
   - for the execution of *i-let* incarnations

2. team
   - of (related/dependent) *i-let* instances
   - for the coordination of invasive processes
Claim Attributes (excerpt)

• coherent/incoherent
• heterogeneous/homogeneous
• preemptive/nonpreemptive
  - temporal, spatial
  - interceptive
• interactive/passive
Team Attributes (excerpt)

- sequential/nonsequential
- coordinated/uncoordinated
- dependent/independent
- clocked/freewheeling
- run to completion/surrender
Principle of Operation

```plaintext
claim = invade(type, quantity, properties);
if (!useful(claim)) {
    /* Retry with alternate claim setting or algorithm. */
}

team = assort(claim, code, data);
if (!viable(team)) {
    /* Retry with alternate team setting or fail. */
}

if (!infect(claim, team)) {    /* employ resource(s) */
    /* Should not happen: Retry later or fail. */
}

retreat(claim);                /* clean-up of resource(s) */
```

---

15
Invasive Sorting

```c
claim = invade(SMP, 42, COHERENT);
if (sizeof(claim) == 1)
    /* only one processing element: sort serial */
else {
    /* 1 < n <= 42 processing elements */

    team = assort(claim, code, data);    /* create workload */
    if (viable(team))
        infect(claim, team);              /* sort in parallel */

    retreat(claim);                     /* await join, clean-up */
}
```
Invasive Ray Tracing

```c
if (all pixels see same object) {
    claim = invade(SIMD, ∞, COHERENT);
    if (useful(claim)) /* n > 1 processing elements */
        team = assort(claim, code(SIMD), data(SIMD));
} else {
    claim = invade(MIMD, 42, COHERENT|HOMOGENEOUS);
    if (useful(claim)) /* 1 < n <= 42 processing elements */
        team = assort(claim, code(MIMD), data(MIMD));
}

if (viable(team))
    infect(claim, team); /* run in parallel */

retreat(claim); /* clean-up */
```
try {
    claim = invade(SMP, ∞, PREEMPTIVE|INTERCEPTIVE);
    /* Virtual claim: assort, infect, and retreat. */
}

catch(int what) {
    switch(what) {
    case CLAIM_CORE: /* Handle request to release core. */
        if (this i-let gets finished shortly)
            allow(remaining period of this i-let);
        else
            yield();
        break;
    /* Other: CLAIM_PAGE, CLAIM_TILE, CLAIM_AREA... */
    }
}
## X10: Language Support

<table>
<thead>
<tr>
<th></th>
<th>C/C++</th>
<th>invasive X10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attributes</strong></td>
<td>argument vector</td>
<td>class hierarchy</td>
</tr>
<tr>
<td><strong>Resource request</strong></td>
<td><code>invasive(...)</code></td>
<td><code>claim.invade(...)</code></td>
</tr>
</tbody>
</table>
| **Resource use** | `infect(...)`       | `intra-place`
|                  | `fan-in`            | `inter-place`
|                  |                     | `async {...} at (...){}{...}` |
| **Resource release** | `retreat(...)`    | `claim.retrofit(...)`      |
| **Critical section** | `blocking`
|                  | `non-blocking`      | `atomic {...}`              |
| **Shared memory** | `tile`              | `place`                    |
System Organization and Operation
MPSoC „InvasIC“
Software Stack

- **C/C++**
- **X10** (application layer)
- **X10 MRT** (run-time layer)
- **libC**
- **OSAL** (operating system layer)
- **agents**
- **OctoPOS**
- **HAL** (hw abstraction layer)
- **LEON**
- **x86**
- **Linux** (hardware layer)
Operating-System Abstraction Layer

- provides an API to an abstract operating system dedicated to invasive computing
- forwards service requests beyond invasive computing to some host operating system
- eases the embedding of invasive application programs into the InvasIC simulation system
- and enhances portability of these programs
Agent System

- takes care of resource allocation on behalf of an invading application program
- different applications compete and cooperate by means of dedicated agents
- agents bargain over incentives and strategies for competition and cooperation
- focus is on latency, energy and utilization...
OctoPOS

Octo — reference to a nature that is:
• highly parallel in its action and
• adaptable to its particular environment

POS (abbr.) for Parallel Operating System:
• an OS that not only aids parallel processing
• but also operates inherently in parallel
OctoPOS Internals
i-let Dispatching

CiC: dynamic core i-let controller
**i-let Dispatching**

1. **create i-let incarnation**

CiC: *dynamic core i-let controller*
i-let Dispatching

1. create i-let incarnation
2. trigger i-let dissemination

CiC: dynamic core i-let controller
i-let Dispatching

1. create i-let incarnation
2. trigger i-let dissemination
3. dispatch i-let to selected core

CiC: dynamic core i-let controller
Data Transfers

active messages resembled
Data Transfers

1. create & pass transfer order

active messages resembled
Data Transfers

1. create & pass transfer order
2. copy (DMA) data to destination

active messages resembled
Data Transfers

1. create & pass transfer order
2. copy (DMA) data to destination
3. disseminate & dispatch i-lets

active messages resembled
Data Transfers

1. create & pass transfer order
2. copy (DMA) data to destination
3. disseminate & dispatch i-lets
4. reuse source buffer & process data

*active messages resembled*
State of Work

Gaisler LEON3 FPGA-based prototyping system:
• coherent, homogeneous, nonpreemptive, passive claim and run-to-completion teams
• 1 compute tile of max. 6 cores

Synopsis CHIPit demonstration system:
• 3x3 array of (compute, memory, I/O) tiles
• max. 24 LEON3 cores

AMD 48-core experimental system: porting...
Systems-Programming Perspective
Field of Interest

- more optimistic – and less pessimistic – concurrency control
- latency hiding through relocation of system activities to available cores
- control of system-level resource sharing among user-level processes
- re-engineering of legacy operating systems for time-sharing or real-time mode
Synchronization
Agile Critical Sections

- procedures vary with degree of contention
- coexisting variants of the same critical region
- "synchronization on the fly"
- hardware-supported (e.g. ASF) or -based

Self-organizing and use-case dependent protection of non-sequential programs against concurrent processes
Exploitation of Cores

- many-core processor
  - universal cores
    - special cores
      - changing core
      - fixed core
        - driver core
          - device core
            - interrupt core
          - service core
            - scheduling core
            - I/O core
            - communication core
            - paging core
            - synchronization core
          - system call handling as a whole
            - system call handling in pieces (pipelined)
            - off-line I/O
            - peripheral control
            - interrupt handling
            - off-line service
            - task delivery
            - file operation
            - protocol stack processing
            - external pager (replacement policy)
            - semaphore/ticket server, RCL
  - load balancing
    - latency hiding (system-inherent activities)
    - system call handling as a whole
    - system call handling in pieces (pipelined)
    - off-line I/O
    - peripheral control
    - interrupt handling
    - off-line service
    - task delivery
    - file operation
    - protocol stack processing
    - external pager (replacement policy)
    - semaphore/ticket server, RCL
The Devil is in the Details

Caches!
Closing
Beyond InvasIC

LAOS (Latency-Aware Operating System)
- latency-awareness in operating systems for massively-parallel processors
- 2 scientists, 2 student research assistants

COKE (Coherency Kernel)
- software-controlled consistency and coherence for many-core architectures
- 2 scientists, 2 student research assistants
Summary

• thousands of processing elements of clustered heterogeneous CPUs need to be engaged

• multiplexing of single cores (then) becomes more and more insignificant

• programming of massively-parallel systems calls for abstraction and resource-awareness

• operating systems have to reflect on these issues and leave old ways behind!