Parallel programming methodologies for manycores

Jeronimo Castrillon
Chair for Compiler Construction (CCC)
TU Dresden, Germany

NeXtream Solution Seminar & Silexica Technology Workshop
October 17, 2018
History and context
The software productivity gap

- Bring complex SW to ever-increasing complex HW
  - Important: Inflection point in comp-arch (2005)

- Pioneering work that led to SLX
  - Auto-parallelization [Ceng08]
  - General software synthesis approach [Castrill11]
  - Mapping and scheduling [Castrill13-13a]
  - Debugging [Castrill11a, Murillo14]
  - Cost modeling [Oden13, Eusse16]
The programming interface continues to broaden as hardware evolves.

- Single-core architectures
- Use multi-core architectures
- Dark Si: specialize
- Post CMOS?

Not only computing, but also memories and interconnect!
Inflection points and programming

- Programming parallel heterogeneous systems
- Domain-specific languages
- The programming interface continues to broaden as hardware evolves
- Tools and methodologies for Post-CMOS systems
- Optimization: Performance, energy efficiency & resilience
Parallel programming
Programming flow: Overview

Application

Architecture model

Analysis

Synthesis

Code generation

MEM subsystem
DMAs, semaphores, PMU
Communication support
VLIW DSP
HW queues
Network Processor
Packet DMA

Non-functional specification

Application Architecture model

Analysis

Synthesis

Code generation

Property models (timing, energy, error, …)

[Castrill13-13a]

PNargs_ifft_r.ID = 6U;
PNargs_ifft_r.PNchannel_freq_coef = f;
PNargs_ifft_r.PNnum_freq_coef = 0U;
PNargs_ifft_r.PNchannel_time_coef = s;
PNargs_ifft_r.channel = 1;
sink_left = IPC1llmrf_open(3, 1, 1);
sink_right = IPC1llmrf_open(7, 1, 1);
PNargs_sink.ID = 7U;
PNargs_sink.PNchannel_in_left = sink_
Pargs_sink.PNnum_in_left = 0U;
PNargs_sink.PNchannel_in_right = sink_
Pargs_sink.PNnum_in_right = 0U;
taskParams.arg0 = (xdc_UArg)&PNargs_s;
taskParams.priority = 1;
ti_sysbios_knl_Task_create((ti_sysbios_knft_Templ, &taskParams, &eb);
glob_proc_cnt++;
hasProcess = 1;
taskParams.arg0 = (xdc_UArg)&PNargs_f;
taskParams.priority = 1;
ti_sysbios_knl_Task_create((ti_sysbios_knft_Templ, &taskParams, &eb);
glob_proc_cnt++;
hasProcess = 1;
taskParams.arg0 = (xdc_UArg)&PNargs_i;
taskParams.priority = 1;
ti_sysbios_knl_Task_create((ti_sysbios_knft_Templ, &taskParams, &eb);
glob_proc_cnt++;
hasProcess = 1;
taskParams.arg0 = (xdc_UArg)&PNargs_s;
taskParams.priority = 1;
Compilation for parallel & heterogeneous systems

- **Understood**
  - Language, compiler and mapping algorithms
  - Hardware modeling, performance estimation
  - Code generation, runtime HW/SW for heterogeneous multicores

- **Current work**
  - Symmetries and language extensions for **scalability**
  - Symmetries and runtimes for **adaptivity**
  - Design centering for **robustness**
Higher-level abstraction for dataflow

- Functional abstraction for implicitly describing the graph
- Not so much about syntax: Clojure, Haskell, Rust, Java, ...

```clojure
(defn translate [server-port]
  (let [[cnn req] (read-socket (accept (open server-port)))]
    [_[_file-name_lang](parse-request req]
    (if (exists? file-name)
      (load-file-from-disk file-name)
      (generate-reply "No such file.")
    )
    (String word (decompose content) ; poor man's translation
      (log "translating word"
        updated-content (collect length (translate word lang))]
      (reply cnn (compose length updated-content)))
```

[Erte18a]

© Prof. J. Castrillon. CCC Research - NeXtream Seminar, 2018
Functional to dataflow

Dataflow Elements:
- $d ::= \vdots$
  - 1-1 node
  - edge
  - port
  - 1-N node
  - N-1 node

Terms:
- $x \mapsto$
- $t \mapsto$
- $(\text{let } [x t] t) \mapsto$
- $(f x) \mapsto$
- $(g x) \mapsto$

variable
lexical scope
apply stateless
function $f$ to $x$
apply stateful
function $g$ to $x$

Dataflow Nodes:
- $\text{not}$
- $\text{true}$
- $\text{false}$
- $\text{ctrl}$
- $\text{sel}$

negation
map to true value
data to control signal
selection

Control Flow:
- $(\text{if } t t t) \mapsto$
- $(\text{seq } t t) \mapsto$

conditionals
sequential evaluation

Predefined Value Functions:
- $\text{len}[]$
- $[] \Rightarrow$
- $\Rightarrow[]$

length of list
list to stream
stream to list

Predefined Functions:
- $(\text{smap } (\text{algo } [x] t) [v_1 \ldots v_n]) \mapsto$
- $(\text{smap } (\text{algo } [x] t) [v_1 \ldots]) \mapsto$

bounded list
unbounded list

[Erte18a]
(ohua :import [web.translation]) ; import the namespace where the used
; functions are defined
(defn translate [server-port]
(ohua (let [[cnn req] (read-socket (accept (open server-port)))
   [ _ file-name _ lang] (parse-request req)
   ['List content length] (if (exists? file-name)
   (load-file-from-disk file-name)
   (generate-reply "No such file.")
   "String word (decompose content) ; poor man's translation
   _ (log "translating word")
   updated-content (collect length (translate word lang))]
   (reply cnn (compose length updated-content)))

Flow:

[Erte18a]

© Prof. J. Castrillon. CCC Research - NeXtream Seminar, 2018
I/O optimization in uServices

- Functional abstraction: amenable for micro-service architectures
- Problem
  - Modularity at odds with performance due to repeated I/O calls
  - Currently solved via complex applicative functors (Facebook)
- Develop simple dataflow rewrites to optimize I/O batching
Second use-case: I/O optimization in uServices

- Functional abstraction: amenable for micro-service architectures
- Develop simple dataflow rewrites to optimize I/O batching

[Diagram showing the number of I/O calls (avg.) vs. number of graph levels for different configurations: haxi, muse, seq, yauhau.]

[Erte18b]

© Prof. J. Castrillon. CCC Research - NeXtream Seminar, 2018
Second use-case: I/O optimization in uServices

- Functional abstraction: amenable for micro-service architectures
- Develop simple dataflow rewrites to optimize I/O batching

![Graph showing service latency vs. number of graph levels]

[haxl (fork) yauhau yauhau (conc I/O)]

Service latency [ms]

# graph levels

0 1000 2000 3000 4000 5000

[Erte18b]

© Prof. J. Castrillon. CCC Research - NeXtream Seminar, 2018
Adaptivity

- Originally in embedded domain: Applications meant to execute alone

- Today
  - Multiple applications sharing resources
  - Available resources unpredictable at load time
  - Design space too large for exploration at running time
  - But: You still want time-predictability

- Strategy
  - Generate multiple (canonical) variants
  - Select and perform cheap transformations at running time

Source: Chen, NTU, MPSoC 2008
Exploiting symmetries

- Intuition
  - SW: Some tasks/processes/actors may do the same
  - HW: Symmetric latencies (CoreX ↔ CoreY)
  - Symmetry: Allows transformations w/o changing the outcome

- No need to analyze all possible mappings (prune search space)

(Symmetries have been implicitly exploited in the past)
Symmetries in Odroid: Example

![Diagram showing mappings and architecture subgraphs]

Equivalent mappings

Graph isomorphism

[Cortex A7, Cortex A15]

[Goens15, Goens17a]
Data-level parallelism: Scalable and adaptive

- Change parallelism from the application specification
- Static code analysis to identify possible transformations (or via annotations)
- Implementation in FIFO library (semantics preserving)

[Diagram showing changes at runtime]

Meta-information & configurations

[Reference: Khasanov18]
Flexible mappings

- Mapping 3
- Mapping 2
- Mapping configuration1

Runtime

Mapping configuration

- Given multiple **canonical** configs by compiler, select one at run-time
- Exploit mapping **equivalences** and **similarities**
Flexible mappings: Tetris with extra rotations

[Goens17b]
Flexible mappings: Run-time analysis

- Modified Linux kernel: symmetry-aware
- Target: Odroid XU4 (big.LITTLE)
- Multi-application scenarios: audio filter (AF) and MIMO
  - 1 x AF
  - 4 x AF
  - 2 x AF + 2 x MIMO
- 3 mappings to two processors
  - T1: Best CPU time
  - T2: Best wall-clock time
  - T3: GBM heuristic

Single AF

[Goens17b]
Flexible mappings: Multi-application results (1)

More predictable performance

Comparable performance to dynamic mapping

[Goens17b]
Flexible mappings: Multi-application results (2)

Better energy predictability as well
Robustness

- Static mappings, transformed or not, provide good predictability
- However: Many things out of control
  - Application data, unexpected interrupts, unexpected OS decisions

Can we reason about robustness of mapping to external factors?
Design centering

- Design centering: Find a mapping that can better tolerate variations while staying feasible.

- Studied field, in e.g., biology, circuit design or manufacturing systems.

- Currently
  - Using a bio-inspired algorithm
  - Robust against OS changes to the mapping
Design centering: Algorithmic

- Intuition: Find the **center** and the **form** of a region, in which parameters deliver a **correct solution**

- Formally
  - $A$: Set of correct solutions
  - $P$: Hitting probability
  - $L$: Generic metric space

- Searching: Allow annealing (dynamically change $P$)

\[
\max_{B=B(m, C) \in L^n} \frac{\text{vol}(B(m, C))}{\text{vol}(B(m, C))} \quad \text{s.t.} \quad m \in A, \quad \frac{\text{vol}(A \cap B(m, C))}{\text{vol}(B(m, C))} \geq P
\]
Evaluation

- Analyze how robust the center really is
  - Perturbate mappings and check how often the constraints are missed
  - Signal processing applications on clustered ARM manycore and NoC manycore (16)

![Graphs showing Mappings passed in % for ARM SoC and NoC Architectures](image)
Evaluation

- Analyze how robust the center really is
  - Perturbate mappings and check how often the constraints are missed
  - Signal processing applications on clustered ARM manycore and NoC manycore (16)

![Graphs showing mappings passed in ARM SoC and NoC architectures.](Hempel17)

Audio-filter
Robustness: SW-based error correction

- Today and future technologies feature hardware faults and soft-errors
  - Need to protect against them at different levels

- Typical approach: Compiler duplicates dataflow and insert checks

Original code

```assembly
%3 = add i64 %0, %1
%4 = mul i64 %3, %2
```

Fault-tolerant code

```assembly
%r3 = add i64 %r0, %r1
%r4 = mul i64 %r3, %r2
%f0 = icmp eq i64 %4, %r4
br i1 %f0, label continue, label recover
```

Source: [Borkar05] [Oh12]
Robustness: AN Encoding

- Arithmetic codes: One can still do meaningful arithmetic on encoded data
- AN encoding: **Make integer values multiples of a fixed constant A**
  - Check for errors like this:
    ```c
    if (n % A != 0) { error_handler(); }
    ```
- Can be automated by compiler!
  - Some operations require non-trivial transformations
  - Integer division: \( m/n \mapsto (A^2m)/(A^n) = A^{n-m} \)
- Advantages over code duplication
  - Data in memory is encoded
  - Good for multithreading and shared memory!
- Disadvantages: Large **runtime overheads** (up to and over several 10x)
Compiler for resilience: Results – Failure rates

[Rink17]
Compiler for resilience: Results – Failure rates

Different optimizations lead to different failure rates
Different optimizations lead to different runtime overheads

Possible direction: Raise the level of abstraction
Domain-specific languages (DSLs)
Domain-specific languages

- Languages evolve, formalizing powerful design patterns (abstractions)
  - Some of them too common, so we do not notice it (goto \(\rightarrow\) structured control, calling conventions \(\rightarrow\) procedures, …)

- DSLs: bridge gap between problem domain and general purpose languages

- Many quite successful DSLs today (dataflow above, also a DSL)
Example: Tensors (Physics and Machine learning)

- Tensor expressions typically occur in numerical codes
  \[ \mathbf{v}_e = (A \otimes A \otimes A) \mathbf{u}_e \]
  - Tensor product: common in computational fluid dynamics
- On performance
  - Matrixes are small, so libraries like BLAS don’t always help
  - Expressions result in deeply nested for-loops
  - Performance highly depends on the shape of the loop nests
- Higher-level expressions: No need for complex polyhedral analysis
Lowering → Tensor IR → Codegen

For $v_e = (A \otimes A \otimes A) u_e$, where $A$ is a matrix and $u$ is a tensorIN:

- **Type Matrix**: $[mp np]$
- **Type TensorIN**: $[np np np ne]$
- **Type TensorOUT**: $[mp mp mp me]$

$\mathbf{v} = \alpha \cdot (\mathbf{A} \odot \mathbf{A} \odot \mathbf{A}) \cdot \mathbf{u}$

### Code Example

```fortran
for (unsigned i0 = 0; i0 < 1000; i0++) {
    double t6[18];
    for (unsigned i3 = 0; i3 < 3; i3++) {
        for (unsigned i2 = 0; i2 < 3; i2++) {
            for (unsigned i1 = 0; i1 < 2; i1++) {
                t6[i1 + 2*(i2 + 3*(i3))] = 0.0;
            }
        }
    }
    for (unsigned i4_contr = 0; i4_contr < 3; i4_contr++) {
        t6[i1 + 2*(i2 + 3*(i3))] = A[i1 + 2*(i4_contr)];
        *u[i2 + 3*(i3) + 3*(i4_contr) + 3*(i0)] = t6[i1 + 2*(i2 + 3*(i3))];
    }
}
```

### Optimizations
- Fortran embedding
- Tensor IR
- Codegen
- Iterative compilation
- Linkable C code
Example: Interpolation operator

- **Interpolation:** \( v_e = (A \otimes A \otimes A) u_e \)

\[
    v_{ijk} = \sum_{l,m,n} A_{kn} \cdot A_{jm} \cdot A_{il} \cdot u_{lmn}
\]

- **Three alternative orders (besides naïve)**
  
  **E1:**

  \[
  v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot (A_{jm} \cdot (A_{il} \cdot u_{lmn})))
  \]

  **E2:**

  \[
  v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot A_{jm}) \cdot (A_{il} \cdot u_{lmn})
  \]

  **E3:**

  \[
  v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot ((A_{jm} \cdot A_{il}) \cdot u_{lmn}))
  \]

[Susungi17, Rink18]
Meta-programming for optimizations: Results (2)

- Extra control allow for new optimization (vs pluto): changing shapes
- General tensor semantics allow covering more benchmarks than TensorFlow

[Susungi18]
Summary

- Current research in tools for heterogeneous manycores
  - High-level abstractions for language scalability
  - Exploit symmetries and variable parallelism for runtime adaptivity
  - Reason about robustness of a mapping and of general code

- Further raise level of abstraction with DSLs
  - Example for tensors (CFD and Machine Learning)
  - Towards more automation on top of adaptive autosar
References


