Compiler Technology for Data-Parallel Languages

Sven-Bodo Scholz

International Summer School on Advances in Programming Languages
Heriot-Watt University
Edinburgh, Scotland

25.-28.August 2009

Multicores are Here!

- Parallelism was
  - academically studied for a few decades
  - affordable only by HPC labs with deep pockets
  - programmed by experts

Multicores are Here!

- Parallelism was
  - academically studied for a few decades
  - affordable only by HPC labs with deep pockets
  - programmed by experts
- Today Parallelism is
  - available cheaply in everybody's PCs and laptops
    - single-core CPUs are history
    - GPGPUs bring hundreds of cores for less than 100 GBP

Multicores are Here!

- Parallelism was
  - academically studied for a few decades
  - affordable only by HPC labs with deep pockets
  - programmed by experts
- Today Parallelism is
  - available cheaply in everybody's PCs and laptops
    - single-core CPUs are history
    - GPGPUs bring hundreds of cores for less than 100 GBP

⇒ Needs to be programmable by general practitioners!

⇒ Opportunity / Obligation for programming language research to provide adequate tools!

The Dawn of a Software Revolution

- Many of the "old truths" do no longer hold!
  - Sequential Truth: redundant computations are evil!
  - Parallel Truth: redundant computation may reduce synchronisation!
  - Sequential Truth: excessive storage use is evil!
  - Parallel Truth: separation of data may eliminate dependencies!

The Dawn of a Software Revolution

- Many of the "old truths" do no longer hold!
  - Sequential Truth: redundant computations are evil!
  - Parallel Truth: redundant computation may reduce synchronisation!
  - Sequential Truth: excessive storage use is evil!
  - Parallel Truth: separation of data may eliminate dependencies!
- Depending on the target hardware we may need to shift between those!

The Dawn of a Software Revolution

- Many of the "old truths" do no longer hold!
  - Sequential Truth: redundant computations are evil!
  - Parallel Truth: redundant computation may reduce synchronisation!
  - Sequential Truth: excessive storage use is evil!
  - Parallel Truth: separation of data may eliminate dependencies!
- Depending on the target hardware we may need to shift between those!

$\Rightarrow$ A declarative approach is needed!

Data-Parallelism

- Fundamental idea:

  Formulate Algorithms in terms of SPACE rather than TIME

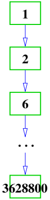
Data-Parallelism

- Fundamental idea:

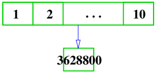
  Formulate Algorithms in terms of SPACE rather than TIME

- Example: factorial

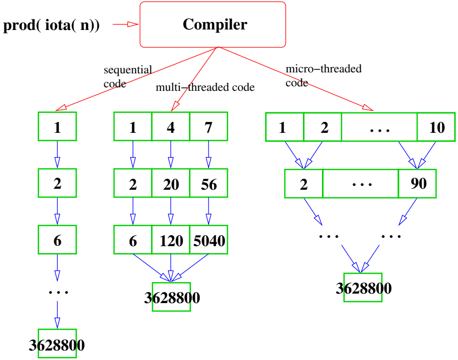
```
prod = 0;
for( i=1; i<=10; i++) {
  prod *= i;
}
```




```
prod( iota( 10))
```



The Compilation Challenge — a first glimpse —



⇒ Different hardware architectures require different code generation strategies!

The Language Challenge

- What data structures are supported?
  - Choice I: homogeneous or inhomogeneous data?
  - Choice II: nested structure or flat?
    - if nested, homogeneously or inhomogeneously?
    - staticly known nesting depth or unlimited nesting?

The Language Challenge

- What data structures are supported?
  - Choice I: homogeneous or inhomogeneous data?
  - Choice II: nested structure or flat?
    - if nested, homogeneously or inhomogeneously?
    - staticly known nesting depth or unlimited nesting?
- What operations are supported?
  - Choice I: map-based only or map-based and fold-based?
  - Choice II: homogeneous or inhomogeneous?
  - Choice III: nested or flat only?

The Language Challenge

- What data structures are supported?
  - Choice I: homogeneous or inhomogeneous data?
  - Choice II: nested structure or flat?
    - if nested, homogeneously or inhomogeneously?
    - staticly known nesting depth or unlimited nesting?
- What operations are supported?
  - Choice I: map-based only or map-based and fold-based?
  - Choice II: homogeneous or inhomogeneous?
  - Choice III: nested or flat only?

⇒ Genericity vs Efficiency Dilemma!

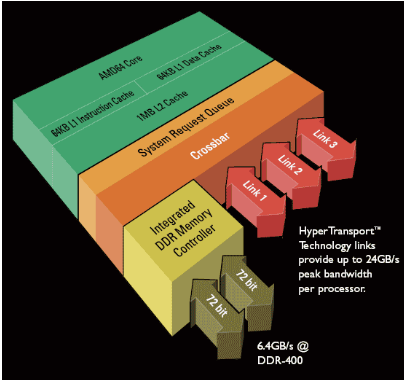
A Collection of Choices Made

- APL / J/ K
- NESL
- SISAL
- Fortran90 / HPF
- SAC
- Google's mapreduce
- Fortress
- data-parallel Haskell

The Compilation Challenge — a second look —

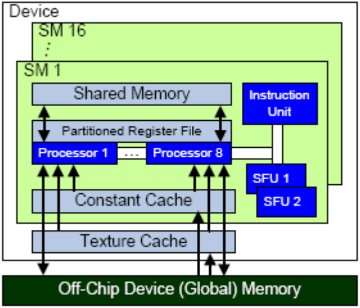
- Hardware and software constraints interfere big time! Examples:
  - Only hohmogeneous data structures benefit from vector instructions!
  - Not all architectures do support truely nested concurrency!
  - Some architectures do not cope well with inhomogeneous operations.
  - Achieving efficient fold operations typically requires architecture dependent measures.
- Getting a single aspect wrong typically is fatal.

Traditional SMPs



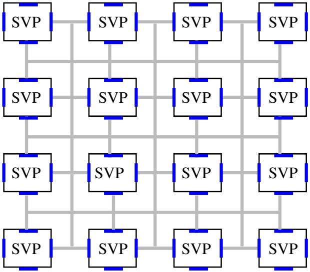
- several standard cores (currently 2-8) on one chip
- thread handling expensive
- synchronisation expensive
- cache coherence expensive
- memory access bottleneck

GPGPUs



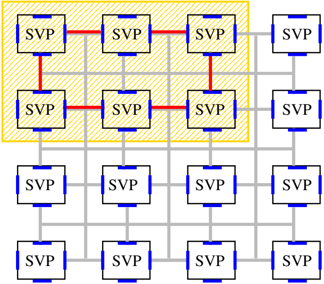
- more than 128 cores
- hardware support for thread creation and synchronisation
- hardware support for thread scheduling
- very restricted thread-functionality
- strictly flat concurrency
- card-private memory

Special Hardware, here: μTC



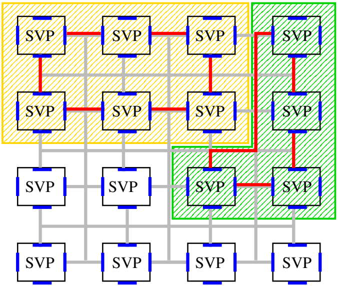
- several hundread full-fledged cores
- hardware support for thread creation
- hardware support for linear synchronisation
- hardware support for thread scheduling
- cash-only memory
- dynamic ressource allocation

Special Hardware, here: μTC



- several hundread full-fledged cores
- hardware support for thread creation
- hardware support for linear synchronisation
- hardware support for thread scheduling
- cash-only memory
- dynamic ressource allocation

Special Hardware, here: μTC



- several hundread full-fledged cores
- hardware support for thread creation
- hardware support for linear synchronisation
- hardware support for thread scheduling
- cash-only memory
- dynamic ressource allocation

Challenge I: Concurrency Overhead Amortisation

typical thread overhead cost (pthreads on solaris):

- thread creation typically several thousands of cycles!
- thread switch costs more than 1000 cycles!
- semaphor-based sync typically as expensive as thread creation!

Measure I: Localise "Thread Management"

Main idea:

create a fixed set of threads, prefereably matching the number of cores available, and have a light-weight solution in the runtime system.

- \+ no OS thread switches needed
- \+ thread creation exactly once upon startup
- \+ lock-free synchronisations
- \+ cheap dynamic scheduling possible
- − potential resource waste

Measure II: Flattening: Maximising Scheduling Flexibility

Main idea:

expose as much concurrency to the local thread management as possible by accumulating nested data parallel situations.

Measure II: Flattening: Maximising Scheduling Flexibility

Main idea:

expose as much concurrency to the local thread management as possible by accumulating nested data parallel situations.

- Blelloch and Sabot, *Compiling Collection-Oriented Languages onto Massively Parallel Computers*, Journal of Parallel and Distributed Computing, 1990.
- Grelck, Scholz and Trojahner, *WITH-Loop Scalarization – Merging Nested Array Operations*, IFL'03, 2004.
- Peyton Jones, Leshchinskiy, Keller and Chakravarty, *Harnessing the Multicores: Nested Data Parallelism in Haskell*, FSTTCS, 2008.

Measure III: Dedicated Hardware Support

Main idea:
novel architectures such as μTC enable more direct exposure of data-parallelism to the hardware.

The overall gain of this approach is in the focus of current research:

`www.apple-core.info`

Challenge II: Computation vs Memory Transfer

Whatever is computed by a single thread on a single node underlies the good "old truths"!
⇒ exessive memory use becomes evil (again)!

Measure I: Transforming Space into Time

Main idea:

use producer / consumer optimisations to avoid data structures to be materialised in memory.

Measure I: Transforming Space into Time

Main idea:

use producer / consumer optimisations to avoid data structures to be materialised in memory.

- Abrams, *An APL Machine*, PhD thesis, 1970.
- Scholz, *With-loop-folding in* SAC*–Condensing Consecutive Array Operations*, IFL'97, 1997.
- Chakravarty and Keller, *Functional Array Fusion*, ICFP'01, 2001.
- Ghuloum, *Ct: C for Throughput Computing* , Intel white paper.
- Russell, Mellor, Kelly and Beckmann, *DESOLA: an Active Linear Algebra Library Using Delayed Evaluation and Runtime Code Generation*, Science of Computer Programming, 2008.

Measure II: Locality Enhancing Scheduling

Main idea:

order the elements computed by a single thread in a cache efficient way.

Measure II: Locality Enhancing Scheduling

Main idea:

order the elements computed by a single thread in a cache efficient way.

- Wolf and Lam, *A Data Locality Optimizing Algorithm*, PLDI'91, 1991.
- Grelck, Kreye and Scholz, *On Code Generation for Multi-Generator WITH-Loops in SAC*, IFL'99, 2000.
- Bondhugula, Hartono, Ramanujam, and Sadayappan, *A practical automatic polyhedral parallelizer and locality optimizer*, PLDI'08, 2008.

Measure III: Latency Hiding

Main idea:

if thread-switches are cheap, we can create several threads on one core!

⇒ non-memory bound computations can hide the memory latency!

Architectures like SUN's Niagra, GPGPUs or μTC benefit directly!

Challenge III: The Aggregate Update Problem

- The data-parallel approach suggests the use of many large data structures.
- It is key to leave it to the compiler to decide which/ how many are being materialised!

⇒ requires a space-efficient implicit memory management!

Measure I: Reference Counting

Main idea:

keep the number of active references to any given data structure in a seperately maintained field.

$\Rightarrow$ enables updates and memory reuse ASAP!

Measure I: Reference Counting

Main idea:

keep the number of active references to any given data structure in a seperately maintained field.

$\Rightarrow$ enables updates and memory reuse ASAP!

- Cann, *Compilation Techniques for High Performance Applicative Computation*, PhD thesis, 1989.
- Trojahner, *Implicit Memory Management for a Functional Array Processing Language*, Diploma Thesis, 2005.

Measure II: Concurrent Heap Management

Main idea:

keep separate heaps for separate threads
⇒ lock-free concurrent memory management can be achieved.

Measure II: Concurrent Heap Management

Main idea:

keep separate heaps for separate threads
$\Rightarrow$ lock-free concurrent memory management can be achieved.

- Berger, Kathryn S. McKinley, Robert D. Blumofe, and Paul R. Wilson, *Hoard: A Scalable Memory Allocator for Multithreaded Applications*, ASPLOS-IX, 2000.
- Joseph Attardi and Neelakanth Nadgir, *A Comparison of Memory Allocators in Multiprocessors*, Sun Developer Network, 2003.
- Grelck and Scholz, *Efficient Heap Management for Declarative Data Parallel Programming on Multicores*, DAMP'08, 2008.

Open Issue: How to deal with dynamic nesting?

- current allocators are mainly effective due to restrictions in the the way threads are created / what threads do
- architectures with hardware support for thread creation / handling break these boundaries
- How do we avoid the re-introduction of lock-based memory operations?

`www.apple-core.info`

Multicores will Enforce a Software Revolution

- Nobody wants to buy a new machine if he does not benefit in terms of performance!
- Hand-parallelising programs is just too hard!

Data-Parallel Programmming defines algorithms in SPACE rather than in TIME

- Data-Parallel Programmming is not just the ability to parallelise loops without dependencies!
- It encourages different program specifications where dependencies are expressed in data rather than time!
- Iterations are expressed as vectors / arrays!
- check it out!

`www.sac-home.org`

Compiling Data-Parallel Programms is Far from Trivial

- all the black-belt knowledge of parallel programming needs to go into the compiler
- getting a seemingly minor detail wrong often prevents from performance gains
- compilion techniques are heavily dependent on the target hardware

Some Solutions Exist

- localised scheduling techniques
- target-dependent space time transformations
- private heap management
- ...
- The techniques shown here enable auto-parallelisation that easily outperforms that of Fortran90/ HPF programs!
- The first autoparallelising compilers for GPGPUs are coming into existance just now!

Much More Work Needs to be Done

- Many new architectures enable new approaches
- How generic can data parallel programs be?
- How can we make use of hybrid architectures?
- Can optimisation happen at runtime?
- ...