At the End of the Last Decade

- In June 2009, ACE was approached by a DVD-chip company to build a compiler for a DSP
  - Functionality \textit{NOT} an issue
  - Delivery time \textit{IS} an issue
  - Assembler, linker and simulator available
- Architecture description was altogether reasonable
- So we accepted the job, estimating 3 person months of work
- So they send the existing tools . . .

The Customer’s Toolset: Much Like this Car

Wrapping up the Project

- Some architectural hiccups
  - Non-interlocking architecture (WTF!?)
  - Different representation for character and integer pointers
  - Required separate control and data stacks
- Took about 4 man-months, one month over budget
  - 50% of the effort on fixing the tools
  - 50% on the compiler
- Compiler delivered before the agreed date
ACE is in the Compiler Business

- Main product: the CoSy compiler development system
  - High quality compiler building tool-set
  - Highly adaptable
  - Aimed at embedded processors
- Motivation: we love the challenges of complex software technology

The Compiler Business is not a Healthy Business

- Order of magnitude: for every 2 Euro compiler development cost, 1 Euro is recovered in sales
- In 2002, we identified 120 companies selling compiler technology
  - Today, 70 of those companies do not exist any more or changed business model

Embedded Industries

- Companies selling things
  - Selling consumer goods, chips or in-between
  - Efficiency of manufacturing is critical
  - Give away toolsets and examples for their products
- Tools companies
  - Like Synopsys
  - Give away designs and libraries to promote tool usage
- IP selling companies (HW and SW)
  - No manufacturing cost (except burning the CD)
  - Example: ARM
  - IP protection by licenses, make money from royalties

Compiler Technology is a Long Term Investment

100% Effort (person-years)

CoSy Performance (beating the theoretical limit)

- STMicroelectronics - FlexCC2 EFR Benchmark on MMDSP+

Regional Culture - A Difficult Sell

- USA:
  - Not invented here, build it from scratch
  - Good at vertical business (Apple, Google power plants)
- Europe:
  - Thorough engineering, but slow
  - Understanding long term investments, also in software
  - Specialized market
- Japan
  - Vertical but no good at software (Cell in PS3)
- India
  - Software mostly, consultancy
- China
  - “Gimme the source, quick!”
  - No intuitive understanding of IP
A Bit of GCC Bashing

Subjectively: hurting ACE in the market
Objectively: hard to understand why it is so popular in research

Common Fallacies

- Santa Claus exists
- The Olympics are about sports
- Benchmarks can be trusted
- Open Source software is cheap
- GCC is a community effort

CoSy Benchmark Figures

Benchmarking CoSy vs GCC compiler for the ARM processor:

CoSy beats GCC by 17%*

"Apps run faster on Windows than Mac OS, generally about 20 percent [because Apple is] using the GCC compiler," says Anup Murarka of Adobe

GCC is the Lowest Common Denominator in Compilers

- Performance lagging 20% compared to dedicated compilers
- Its software engineering structure is negligible
- It is targeted to architectures that are 20+ years old

So, GCC is unsuitable for research, however....

Regional Culture

Researchers are Like the Chinese

- USA:
  - Not invented here, build it from scratch
  - Good at vertical business (Apple, Google power plants)
- Europe:
  - Thorough engineering, but slow
  - Understanding long term investments, also in software
  - Specialized market
- Japan
  - Vertical but no good at software (Cell in PS3)
- India
  - Software mostly, consultancy
- China
  - "Gimme the source, quick!
  - No intuitive understanding of IP
  - Much like the average researcher

Compiler trouble?

Dial +31(0)20-6646416 for Help

ACE: your highway to compiler bliss
The Next Decade in Compiler Technology:

Compiling for Parallel Architectures

---

The Next Decade

- Building dumb-transparent compilers
  - Explicitly parallel programming
  - Maps programs to the specific architecture they were programmed for
  - No (performance) portability
  
or

- Parallelism mapping compilers
  - Map parallelism in the application to a variety of (current and future) architectures
  - The road to portability

---

The Case for the Dumb-Transparent Compiler

- One CoSy user on the brink of achieving world domination in LTE
- Using a vector architecture
- Use the compiler to do register allocation, scheduling and software pipelining
- Use intrinsics for vector operations
- Turned off almost all optimizations

*Because 90% performance is not good enough*

---

Needed in any case: Representation of Parallel Models

- The task of the IR is to facilitate the mapping to the architecture
- So, the architecture models need to be present as programming models in the IR
- The IR must accommodate step-wise refinement, possibly with user intervention

---

New Parallel Languages?

*In the embedded industry, C will remain dominant for the next 10 years*

- Industry is conservative, evolutionary
  - Even C++ has too much overhead
- X10, Chapel, Fortress?
- Too many existing C programs and programmers
- There will be variants: OpenMP, OpenCL

- CoSy’s flexible architecture can accommodate such variants

---

Why so Difficult?

- Concurrency is a feature
  - Allows decoupling of I/O(GUI) and computation

- Parallelism is a limitation of the architecture
  - Does not add functionality
  - Makes programming more difficult
  - Any overhead will cost performance
New Parallel Architectures? It’s a Mess

- No dominant parallel architecture in the nearby future
- IBM Cell Architecture 2001-2009†
- Intel Larrabee 2007-2009†
- That is a bit of a problem for us, compiler builders....

Order in Chaos

Coarse Grain | Multi-Thread | Grid
---|---|---
Heterogeneous
Heterogeneous
Homo-
Fine Grain

Observation: there are only 4 Parallel Architecture Models

- Heterogeneous multi-core
- Grid
- Vector
- Multi-threading

Architecture Characterization Hierarchy

Data distribution, determines SPMD code running on each node

Vectorization, based on attributes like restrict and alignment

Compiler can handle that

Heterogeneous Multi-Core (HMC)

- RISC + DSP = MPSoC
- Power may be more important than parallelism
- Program code needs to be split
- Data follows code, possibly migrating dynamically

Grid Architectures

- Distributed memory
- Data Distribution (HPF-like) is key step
- Communication follows from data distribution
Vector Architectures

- Register to Register Vector parallel operations
- Single Program Counter
- Exploits fine-grain inner loop parallelism
- Alignment and cache layout data requirements

Shared Memory Multi-Threading Architectures

- Shared memory multi-processing (OpenMP)
- Tasks may be specialised (heterogeneous)
- Hardware determines lower bound on grain size

Covering the Parallel Universe

- Multi-core data parallelism
- Vector ILP

Semi-Automatic Parallelizing Framework

- C with high level parallel extensions
- Colby Compiler: Parallelism = CCMIR+HMC+Grid+Threading+Vector

Key Technologies

- Whole Program Analysis
- Loop nest analysis and transformation

Whole Program Analysis

- Full understanding of the program flow/call graph
- Full understanding of the data flow
  - Alias tracking, points-to analysis
- Must allow for the interface definition of “black boxes”
- Transform the program using region selection (for heterogeneous mapping)
  - Define the region interface - data and control; same format as for black boxes
Nested Loop Parallelism

- The compiler’s Swiss army knife
- Dependencies can be analyzed automatically using ILP techniques
- Can be mapped to SIMD, Vector, Stream, Grid and multi-threaded architecture

A Research Question on Memory Consistency Models and Optimization

Discussing a little known secret

Memory Consistency Models

- The Memory Consistency Model defines how one thread sees the shared memory in another thread
- Two main models:
  - Sequential consistency
  - Relaxed consistency

Sequential Consistency: A Implementation’s Straitjacket

Sequential Consistency: A Possible Interleaving
Sequential Consistency does not Permit Caching

\[ A = B = 0 \]

\[ T_1 \]

\[ = B \]
\[ A = 1 \]
\[ X = B \]

\[ = A \]
\[ B = 1 \]
\[ Y = A \]

\[ T_2 \]

Must have:
\[ X = 1 \land Y = 1 \]

Control Synchronization

Relaxed Consistency: Allows Caching!

\[ A = B = 0 \]

\[ T_1 \]

\[ = A \)
\[ A = 1 \]
\[ x = B \]

\[ = B \]
\[ B = 1 \]
\[ y = A \]

\[ T_2 \]

\[ Allowed: x = 0 \land y = 0 \]

Synchronize & Flush

\[ A = B = 0 \]

\[ x = B \]
\[ A = 1 \]

\[ y = A \]
\[ B = 1 \]

\[ x = 0 \land y = 0 \]

Synchronize & Flush

Definition: Race Condition

- A race condition occurs when there is an unsynchronized
  - Read-Write or
  - Write-Write
  of a shared variable

Without Race Conditions, Semantics is Easy

\[ \neg G \]
\[ G = 42 \]

assert( G == 42 )

Theorem Linking Relaxed and Sequential Consistency

A correctly synchronized program [i.e. no race conditions] under relaxed memory consistency behaves equal to its sequential consistent implementation

Little Known Secret

Valid sequential optimizations can introduce writes where none existed

```c
workerThread() {
    workerThread() {
        reg = sharedV;
    for(...) {
        ... work ...
        if( found ) {
            sharedV = result;
            reg = result;
        }
    }
}
}
sharedV = reg;
```
Sequential Optimizations can Create Race Conditions

G = 42

Questions

- What sequential optimizations are invalid for parallel code?
- How can we identify such optimizations?
- How can we test optimizations for this?

Memory Consistency

Background Literature

- “Threads Cannot be Implemented as a Library”
  - Boehm, Hans-J.
  - Google for: “omp memory consistency contradiction”
    will get you to the mailing list discussion