Berkeley UPC Compiler Status Report
and Future Plans

Costin Iancu

LBNL
• There were one...

• There is two now!

• Meet **Seung-Jai Min**...
  
  • OpenMP, GPU, DSMs
Overview

• Now that you’ve met Seung-Jai...

• Compiler Status

• Future Plans
  • Language evolution
  • UPC for multicore
  • Search based optimizations
• Programs need to be efficient and have scalable performance: *performance portability* – focused on latency hiding in clusters

• Manual optimizations for communication: *hard*

• Requirements for performance:
  • Selection of best communication primitives
  • Selection of best transformations for overlap

• Source level optimizations:
  • Message Vectorization - selection of communication primitives
  • Message Strip-Mining - decomposition and scheduling
  • Data Redistributions – scatter/gather on multicore clusters
Challenges

- Performance depends on the application and system characteristics
- Need methodology for compile time and on-the-fly optimizations
  - Robust in the presence of imprecision
  - Lightweight
  - And that works too!
- Adaptation (compiler/static optimizations are not sufficient)
  - Program Description
  - Parameter Estimation/Classification
  - Feedback Loop
  - Performance Models vs Autotuning
    - Guided vs Automatic
    - Instantaneous vs Asymptotic
    - Offline vs Online
BUPC Infrastructure

- Compile/runtime time analysis and transformations
  - Determine program behavior (semantic, performance)
  - Describe behavior to runtime (discrete, coarse grained approximations)
  - Runtime analysis for adaptation, best implementation

- Summarize memory access pattern, lightweight (Paek - LMAD perfect nests)
  - RT-LMAD similar to SSA- irregular loops
  - Easily extended for symbolic analysis

- Multi-versioning: code templates, instantiated and selected at runtime

- Performance models, access control

- Decouple serial transformations from communication transformations
  - Serial transformations - cache parameters (static/conservative)
  - Communication transformations - network parameters (dynamic)

No performance loss when decoupling
(compute/comm intrinsic ratio + multithreading)
Compile Time Transformations

Communication Oblivious Loop Transformations

Communication Aware Loop Analysis

Message Vectorization
Message Strip-Mining
Data Redistribution

Estimation of Loop Overhead

Loop Nest Descriptor + Template Loop Code

Runtime Mechanisms

Estimate Loop Body Overhead

Analyze Comm Requirements

Estimate Load

Instantiate Comm Plan

Eliminate Redundant Comm & Reshape

Performance Database

Performance Models

Memory Manager (Cache)
\textbf{Example: Generated Code}

\begin{verbatim}
for(i=0;i<N;i++)
    sumv += base[i];

ln = start_nest(key);
add_polytope_dim(ln, DEPTH, LB, UB, STRIDE);
br = new_base_ref(ln, ALIAS, element_size;
lmad = new_lmad(ln, br, base_ptr, READ);
add_sos_dim(ln, br, lmad, 0, stride, span);
refvect = analyze_transfers(ln);
if(refvect==1) {
    lbase = (double *) get_local_address(ln, br, lmad);
    sd = get_strips(ln);
    for(oidx = 0; oidx <= ((N-1/sd) -1); oidx = oidx + 1) {
        advance_dim(ln, DEPTH);
        for(iidx = 0; iidx <= (sd -1); iidx = iidx + 1) {
            i = iidx + oidx * sd;
            sumv = lbase[i] + sumv;
        }
    }
}
else {
    //fallback code - shared memory version
    end_nest(ln);
}
\end{verbatim}

\begin{itemize}
    \item \textbf{Describe communication requirements}
    \item \textbf{Simplify, apply model, compute schedule, instantiate communication}
    \item \textbf{Memory management, data cache}
    \item \textbf{Choose communication granularity}
    \item \textbf{Start and retire communication ops}
\end{itemize}
• Compiler released at SC’08
  • Productized and released optimizations for fine-grained programs:
    split-phase, access coalescing
  • Beta release of “vectorization”
  • Papers at ICS’08, PPoPP’08

• The performance model based approach works WELL, but:
  • It requires Costin Iancu to tune it, it requires a PhD in CS to understand it
  • Any change in the system software changes the outcomes

• It’s not like autotuning will automagically fix this problem
  • We know how to autotune 1 thread per core in dedicated environments (see 2nd talk)
  • Noise, flow control
  • Cross-compilation

• Need automated parameter discovery, validation, feedback mechanisms (applies both to performance models and autotuners)
  • A lot of research and engineering work
Language Evolution

• We have non-blocking extensions, Vector/Index/Strided, non-blocking collectives, team collectives – *library extensions for now*

• Language adoption will require compiler integration

• **Data layout is static** in current spec: programmability problems

• We want UPC to be one the languages for programming multicore clusters (it is a very good research platform)

• **Multi-level optimizations (distributed/shared memory/hybrid) : next**

• Extensions required:
  • Dynamic parallelism
  • More powerful data distributions mechanisms (see UPC on GPUs talk, IBM)
Leverage existing research in shared memory programming

Interoperate with other successful candidates

Possible path: integrate with the OpenUH OpenMP compiler
  - Open64 code base
  - OpenMP widely used and supported by industry
  - Will allow us to experiment with
    - Interoperability with other runtimes
    - Dynamic tasking
    - SPMD + work stealing
    - Hybrid programming models
    - Propose UPC language extensions

Approached UH – both very interested in collaboration
Search Based Optimizations

• Current big research and technology push for “smarter” search based compilers
  • DARPA $16M: PACE platform-aware compilation environment
  • Milepost GCC: interactive machine-learning-enabled research tool suitable for adaptive computing (EU effort)

• The focus of these efforts is multi-socket programming

• Our “mission” – large scale multicore clusters

• Challenges
  • Cross compilation
  • Noise, dynamic behavior (previous work gave us a lot of insight)

• Path forward:
  • GCC-UPC and Milepost
  • Talk to PACE folks, extend Open64 (btw very active community)
Infrastructure and Interoperability

- Pathscale just released their code base
- Integrate with new Open64 (OpenUH+ Pathscale + BUPC)
- Extend the existing optimization infrastructure to handle whole program optimization (IPA)
- Extend analysis infrastructure to work with imperfect loop nests and loops with conditionals
- Interoperability and optimizations
  - Pboost library with C++?
  - Scripting languages?
Thank You!