Adaptive Scientific Software Libraries

Lennart Johnsson

Advanced Computing Research Laboratory
Department of Computer Science
University of Houston
Challenges

• Diversity of execution environments
  – Growing complexity of modern microprocessors.
    • Deep memory hierarchies
    • Out-of-order execution
    • Instruction level parallelism
  – Growing diversity of platform characteristics
    • SMPs
    • Clusters (employing a range of interconnect technologies)
    • Grids (heterogeneity, wide range of characteristics)

• Wide range of application needs
  – Dimensionality and sizes
  – Data structures and data types
  – Languages and programming paradigms
Challenges

• Algorithmic
  – High arithmetic efficiency
    • low floating-point v.s. load/store ratio
  – Unfavorable data access patterns (big $2^n$ strides)
    • Application owns the datastructures/layout
  – Additions/multiplications unbalanced

• Version explosion
  – Verification
  – Maintenance

Texas Learning & Computation Center
Opportunities

- Multiple algorithms with comparable numerical properties for many functions
- Improved software techniques and hardware performance
- Integrated performance monitors, models and data bases
- Run-time code construction
Approach

- Automatic algorithm selection – polyalgorithmic functions (CMSSL, FFTW, ATLAS, SPIRAL, …..)
- Exploit multiple precision options
- Code generation from high-level descriptions (WASSEM, CMSSL, CM-Convolution-Compiler, FFTW, UHFFT, …..)
- Integrated performance monitoring, modeling and analysis
- Judicious choice between compile-time and run-time analysis and code construction
- Automated installation process
The UHFFT

- Program preparation at installation (platform dependent)
- Integrated performance models (in progress) and data bases
- Algorithm selection at run-time from set defined at installation
- Automatic multiple precision constant generation
- Program construction at run-time based on application and performance predictions

Texas Learning & Computation Center
Performance Tuning Methodology

**Input Parameters**
- System specifics
- User options

**Installation**
- UHFFT Code generator
- Library of FFT modules
- Performance database

**Run-time**
- **Input Parameters**
  - Size, dim., ...
- **Initialization**
  - Select best plan (factorization)
- **Execution**
  - Calculate one or more FFTs
- **Performance Monitoring**
  - Database update

**Performance Tuning**
- Methodology
  - Library of FFT modules
  - Performance database
  - User options

*Texas Learning & Computation Center*
The UHFFT Software Architecture

UHFFT Library

- Library of FFT Modules
- Initialization Routines
- Execution Routines
- Utilities
  - FFT Code Generator
  - Mixed-Radix (Cooly-Tukey)
  - Prime Factor Algorithm
  - Split-Radix Algorithm
  - Rader's Algorithm

Unparser
Scheduler
Initializer (Algorithm Abstraction)
Optimizer

Key:
- Fixed library code
- Generated code
- Code generator
The UHFFT: Code Generation

- Structure
  - Algorithm abstraction
  - Optimization
  - Generation of a DAG
  - Scheduling of instructions
  - Unparsing

- Implementation
  - Code generator is written in C
    - Speed, portability and installation tuning
  - Highly optimized straight line C code
  - Generates FFT codelets of arbitrary size, direction, and rotation

*Texas Learning & Computation Center*
The UHFFT: Code Generation (cont’d)

- Basic structure is an *Expression*
  - Constant, variable, sum, product, sign change, …
- Basic functions
  - Expression sum, product, assign, sign change, …
- Derived structures
  - Expression vectors, matrices and lists
- Higher level functions
  - Matrix vector operations
  - FFT specific operations
- Algorithms currently supported
  - Rader (two versions), PFA, Split-radix, Mixed-radix
The UHFFT: Code Generation Mixed-Radix Algorithm

Equation: \( W_n = (W_r \otimes I_m)D_{r,m}(I_r \otimes W_m)I_{n,r} \)

Is implemented as:

```c
/**
 * FFTMixedRadix() Mixed-radix splitting.
 * Input:
 *      r       radix,
 *      dir, rot direction and rotation of the transform,
 *      u       input expression vector.
 */
ExprVec *FFTMixedRadix(int r, int dir, int rot, ExprVec *u)
{
    int m, n = u->n, *p;
    m = n/r;
p = ModRSortPermutation(n, r);
u = FFTxI(r, m, dir, rot,
        TwiddleMult(r, m, dir, rot,
            IxFFT(r, m, dir, rot, PermuteExprVec(u, p)))));
free(p);
return u;
}
```
The UHFFT: Performance Modeling

- Analytic models
- Cache influence on library codes
- Performance measuring tools (PCL, PAPI)
- Prediction of composed code performance
- Updated from execution experience

- Data base
  - Library codes. Recorded at installation time
  - Composed codes. Recorded and updated for each execution.

Texas Learning & Computation Center
The UHFFT:
Execution Plan Generation

- Optimal plan search options
  - Exhaustive
  - Recursive
  - Empirical

- Algorithms used
  - Rader (FFTW, UHFFT)
  - PFA (UHFFT)
  - Split-radix (UHFFT)
  - Mixed-radix (FFTW, SPIRAL, UHFFT)
## Characteristics of Some Processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock frequency</th>
<th>Peak Performance</th>
<th>Cache structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Pentium IV</td>
<td>1.8 GHz</td>
<td>1.8 GFlops</td>
<td>L1: 8K+8K, L2: 256K</td>
</tr>
<tr>
<td>AMD Athlon</td>
<td>1.4 GHz</td>
<td>1.4 GFlops</td>
<td>L1: 64K+64K, L2: 256K</td>
</tr>
<tr>
<td>Intel Itanium</td>
<td>800 Mhz</td>
<td>3.2 GFlops</td>
<td>L1: 16K+16K, L2: 92K, L3: 2-4M</td>
</tr>
<tr>
<td>IBM Power3/4</td>
<td>375 MHz</td>
<td>1.5 GFlops</td>
<td>L1: 64K+32K, L2: 1-16M</td>
</tr>
<tr>
<td>HP PA 8x00</td>
<td>750 MHz</td>
<td>3 GFlops</td>
<td>L1: 1.5M + 0.75M</td>
</tr>
<tr>
<td>Alpha EV67/68</td>
<td>833 MHz</td>
<td>1.66 GFlops</td>
<td>L1: 64K+64K, L2: 4M</td>
</tr>
<tr>
<td>MIPS R1x000</td>
<td>500 MHz</td>
<td>1 GFlop</td>
<td>L1: 32K+32K, L2: 4M</td>
</tr>
</tbody>
</table>
Codelet efficiency

Intel PIV 1.8 GHz

AMD Athlon 1.4 GHz

PowerPC G4 867 MHz

Texas Learning & Computation Center
Radix-4 codelet efficiency

Intel PIV 1.8 GHz
AMD Athlon 1.4 GHz
PowerPC G4 867 MHz

Texas Learning & Computation Center
Radix-8 codelet efficiency

Intel PIV 1.8 GHz
AMD Athlon 1.4 GHz
PowerPC G4 867 MHz

Texas Learning & Computation Center
UHFFT Performance comparison for 32 bit processors

- Intel Pentium 4 1.8 GHz
- AMD Athlon 1.4 GHz
- PowerPC G4 867 MHz

"MFLOPS"

Transform size
Power3 plan performance

222 MHz
888 Mflops

Texas Learning & Computation Center
Power3 plan performance

PFA sizes

Transform size

800 Mflops peak

Texas Learning & Computation Center
<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock frequency</th>
<th>Peak Performance</th>
<th>Cache structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Itanium</td>
<td>800 Mhz</td>
<td>3.2 GFlops</td>
<td>L1: 16K+16K (Data+Instruction)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: 92K, L3: 2-4M (off-die)</td>
</tr>
<tr>
<td>Intel Itanium 2</td>
<td>900 Mhz</td>
<td>3.6 GFlops</td>
<td>L1: 16K+16K (Data+Instruction)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: 256K, L3: 1.5M (on-die)</td>
</tr>
<tr>
<td>Intel Itanium 2</td>
<td>1000 Mhz</td>
<td>4 GFlops</td>
<td>L1: 16K+16K (Data+Instruction)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: 256K, L3: 3M (on-die)</td>
</tr>
<tr>
<td>Sun UltraSparc-III</td>
<td>750 Mhz</td>
<td>1.5 GFlops</td>
<td>L1: 64K+32K+2K+2K (Data+Instruction+Pre-fetch+Write)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: up to 8M (off-die)</td>
</tr>
<tr>
<td>Sun UltraSparc-III</td>
<td>1050 Mhz</td>
<td>2.1 GFlops</td>
<td>L1: 64K+32K+2K+2K (Data+Instruction+Pre-fetch+Write)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: up to 8M (off-die)</td>
</tr>
</tbody>
</table>
### Memory Hierarchy

<table>
<thead>
<tr>
<th></th>
<th>Itanium-2 (McKinley)</th>
<th>Itanium</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size:</strong></td>
<td>16KB + 16KB</td>
<td>16KB + 16KB</td>
</tr>
<tr>
<td><strong>Line size</strong></td>
<td>64B/4-way</td>
<td>32B/4-way</td>
</tr>
<tr>
<td><strong>Associativity</strong></td>
<td>64B/4-way</td>
<td>1 cycle</td>
</tr>
<tr>
<td><strong>Latency:</strong></td>
<td>1 cycle</td>
<td>1 cycle</td>
</tr>
<tr>
<td><strong>Write Policies:</strong></td>
<td>Write through, No write allocate</td>
<td>Write through, No write allocate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>256KB</th>
<th>96K B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Line size</strong></td>
<td>128B/8-way</td>
<td>64B/6-way</td>
</tr>
<tr>
<td><strong>Associativity</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Integer Latency:</strong></td>
<td>Min 5 cycles</td>
<td>Min 6 cycles</td>
</tr>
<tr>
<td><strong>FP Latency:</strong></td>
<td>Min 6 cycles</td>
<td>Min 9 cycles</td>
</tr>
<tr>
<td><strong>Write Policies:</strong></td>
<td>Write back, write allocate</td>
<td>Write back, write allocate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>3MB or 1.5MB on chip</th>
<th>4MB or 2MB off chip</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Line size</strong></td>
<td>128B/12-way</td>
<td>64B/4-way</td>
</tr>
<tr>
<td><strong>Associativity</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Integer Latency:</strong></td>
<td>Min 12 cycles</td>
<td>Min 21 cycles</td>
</tr>
<tr>
<td><strong>FP Latency:</strong></td>
<td>Min 13 cycles</td>
<td>Min 24 cycles</td>
</tr>
<tr>
<td><strong>Bandwith:</strong></td>
<td>32B/cycle</td>
<td>16B/cycle</td>
</tr>
</tbody>
</table>

*Texas Learning & Computation Center*
# Itanium Comparison

<table>
<thead>
<tr>
<th>Workstation</th>
<th>HP i2000</th>
<th>HP zx2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>800 MHz Intel Itanium</td>
<td>900 MHz Intel Itanium 2 (McKinley)</td>
</tr>
<tr>
<td>Bus Speed</td>
<td>133 MHZ</td>
<td>400 MHz</td>
</tr>
<tr>
<td>Bus Width</td>
<td>64 bit</td>
<td>128 bit</td>
</tr>
<tr>
<td>Chipset</td>
<td>Intel 82460GX</td>
<td>HP zx1</td>
</tr>
<tr>
<td>Memory</td>
<td>2 GB SDRAM (133 MHz)</td>
<td>2 GB DDR SDRAM (266 MHz)</td>
</tr>
<tr>
<td>OS</td>
<td>64-bit Red Hat Linux 7.1</td>
<td>HP version of the 64-bit RH Linux 7.2</td>
</tr>
<tr>
<td>Compiler</td>
<td>Intel 6.0</td>
<td>Intel 6.0</td>
</tr>
</tbody>
</table>
HP zx1 Chipset

Features:
• 2-way and 4-way
• Low latency connection to the DDR memory (112 ns)
  • Directly (112 ns latency)
  • Through (up to 12) scalable memory expanders (+25 ns latency)
• Up to 64 GB of DDR today (256 in the future)
• AGP 4x today (8x in the future versions)
• 1-8 I/O adapters supporting
  • PCI, PCI-X, AGP
UHFFT Minimum Forward Codelet Performance

- Intel Itanium 800 MHz
- Intel Itanium 2 900 MHz
- Sun UltraSparc-III 750 MHz

MFlop/s vs Codelet size

Texas Learning & Computation Center
Codelet Performance Radix-2

Texas Learning & Computation Center
Codelet Performance Radix-3

UHFFT Radix-3 Itanium 800 MHz, Pavg = 382.0

UHFFT Radix-3 UltraSparc III 750 MHz, Pavg = 296.2

UHFFT Radix-3 Itanium 2 (McKinley) 900 MHz, Pavg = 491.5

Texas Learning & Computation Center
Codelet Performance Radix-4

Texas Learning & Computation Center
Codelet Performance Radix-5

Texas Learning & Computation Center
Codelet Performance Radix-6

Texas Learning & Computation Center
Codelet Performance Radix-7

Texas Learning & Computation Center
Codelet Performance Radix-8

Texas Learning & Computation Center
Codelet Performance Radix-9

Texas Learning & Computation Center
Codelet Performance Radix-10

Texas Learning & Computation Center
Codelet Performance Radix-11

Texas Learning & Computation Center
Codelet Performance Radix-12
Codelet Performance Radix-13

UHFFT Radix-13 Itanium 900 MHz, Pavg = 651.9

UHFFT Radix-13 UltraSparc III 750 MHz, Pavg = 119.5

UHFFT Radix-13 Itanium 2 (McKinley) 900 MHz, Pavg = 1311.6

Texas Learning & Computation Center
Codelet Performance Radix-14

Texas Learning & Computation Center
Codelet Performance Radix-15

Texas Learning & Computation Center
Codelet Performance Radix-16

Texas Learning & Computation Center
Codelet Performance Radix-17

Texas Learning & Computation Center
Codelet Performance Radix-18

UHFFT Radix-18 Itanium 900 MHz, Pavg = 770.1

UHFFT Radix-18 Itanium 2 (McKinley) 900 MHz, Pavg = 1460.3

UHFFT Radix-18 UltraSparc III 750 MHz, Pavg = 245.8

Texas Learning & Computation Center
Codelet Performance Radix-19
Codelet Performance Radix-20

Texas Learning & Computation Center
Codelet Performance Radix-21

Texas Learning & Computation Center
Codelet Performance Radix-22

Texas Learning & Computation Center
Codelet Performance Radix-23

Texas Learning & Computation Center
Codelet Performance Radix-24
Codelet Performance Radix-25

Texas Learning & Computation Center
Codelet Performance Radix-32
Codelet Performance Radix-36

UHFFT Radix-36 Itanium 800 MHz, Pavg = 815.9

UHFFT Radix-36 UltraSparc III 750 MHz, Pavg = 279.1

UHFFT Radix-36 Itanium 2 (McKinley) 900 MHz, Pavg = 1554.6

Texas Learning & Computation Center
Codelet Performance Radix-45
Codelet Performance Radix-64

Texas Learning & Computation Center
The UHFFT: Summary

- Code generator written in C
- Code is generated at installation
- Codelet library is tuned to the underlying architecture
- The whole library can be easily customized through parameter specification
  - No need for laborious manual changes in the source
  - Existing code generation infrastructure allows easy library extensions
- Future:
  - Inclusion of vector/streaming instruction set extension for various architectures
  - Implementation of new scheduling/optimization algorithms
  - New codelet types and better execution routines
  - Unified algorithm specification on all levels
New Tools for Library Code Development

- Generalization of the tools developed for the UHFFT library
- CODELAB: A Developers' Tool for Efficient Code Generation and Optimization
- Combination of
  - High-level scripting language
  - Code generator
  - Performance measurement tools
  - Visualization
- Under development
- Several test examples show very promising results
CODELAB IDE STRUCTURE

CODELAB IDE

Script Interpreter → Code Generator → Visualization → Performance measurement

User Input → Library code → Support Code → Application

Compiler → Execution

Operating System

Texas Learning & Computation Center
CODELAB Structure

- Application consists of
  - Library code:
    - Automatically generated and optimized collection of subroutines
  - Supporting code:
    - Code that binds the library routines together
    - It could be hand-written or automatically generated
  - Application is instrumented for performance measurements automatically
CODELAB Structure

- User writes:
  - Simple script that produces the code generator or supporting code
  - Supporting code for the application
  - The code generator should be able to produce a large variety of code depending on a few input parameters (otherwise it is simpler to write the code by hand)
  - Example: A single code generator for FFT codelets of different size, type, direction, rotation, …

- Script Interpreter:
  - Simplifies construction of the code generator
  - Very restricted set of commands at the moment

Texas Learning & Computation Center
CODELALB Structure

- Code generator
  - C program that generates the application and supporting code
  - Uses abstract expression algebra for code generation
  - Several layers of software:
    - Basic expressions
    - Complex expressions algebra
    - Vector and matrix algebra
    - Polynomial algebra
  - The generated code can be instrumented for performance measurements
  - The initial expression list is transformed into DAG and optimized:
    - Simplification of expressions
    - Folding of constants
  - User can get a variety of information about the generated code:
    - Number of arithmetic ops
    - DAG graph, etc.
CODELAB Structure

- Performance measurements
  - The application can be compiled and executed from within the IDE
  - The performance data are collected and visualized
  - User can modify the code and repeat the process until a satisfactory performance is obtained
  - Detailed performance information by using PAPI library interface
CODELAB Applications

- FFT and DSP Libraries
- Efficient multiple precision arithmetic
- Finite Element Methods
- Linear Algebra
- Other well structured applications that allow for simple parameterization
  - few parameters define a large variety of code
Overview

• UHFFT Performance on some new architectures
  – Intel Itanium 800 MHz,
  – Intel Itanium 2 (McKinley) 900 MHz
  – Sun UltraSparc-III 750 MHz

• New Tools for Library Code Development
  – CODELAB Integrated Development Environment (IDE)
    • Introduction
    • Structure of the CODELAB IDE
    • Applications
The UHFFT: An Adaptive FFT Library

- UHFFT employs more ways of combining codelets for execution than any other library
- Better coverage of the space of possible algorithms
- The PFA algorithm yields good performance where the Mixed-Radix algorithm (MR) performs poorly
  - PFA algorithm requires less FP operations than MR
  - Data access pattern in PFA is more complex than in MR, but large $2^n$ strides can be avoided
- Example IBM Power3
  - Good: 128-way set associative L1 data and instruction caches
  - Bad: Direct mapped L2 cache very vulnerable to cache trashing despite the large cache size
  - PFA execution model works better for large FFT sizes
Acknowledgements

Contributors

Dragan Mirkovic
Rishad Mahasoom
Fredrick Mwandia
Nils Smeds

Funding:

Alliance (NSF)
LACSI (DoE)