High Performance DoD DSP Applications

Robert Bond
Embedded Digital Systems Group
MIT Lincoln Laboratory
23 August 2003
Outline

- DoD High-Performance DSP Applications
- Middleware (with some streaming constructs)
- Future Directions
- Summary
Embedded Signal Processing Requirements

EMBEDDED PROCESSING REQUIREMENTS WILL REQUIRE TFOPS IN THE 2005-2010 TIME FRAME
Embedded Processing Spectrum
Applications vs. Digital Technology

Programmable Processors | Reconfigurable Computing With FPGAs | ASIC - Standard Cell (Conventional Packaging) | Full Custom VLSI Multi-Chip Modules

- Space
- Seeker
- UAV
- Airborne
- Shipboard
- Small Unit Operations
- SIGINT

- High performance Programmable Signal Processors (PSP) address a wide range of applications
- Efficient Stream Processing is the challenge!
Radar Stream Signal Processing Algorithms

GMTI (STAP) Algorithm

SAR Algorithm

Synthetic Aperture Radar (SAR): ~ 100-1000 GOPS ~ 1-100 s latency
Ground Moving Target Indicator (GMTI) ~ 100-1000 GOPS ~ 0.1-1 s latency
Space-Time Adaptive Processing (STAP) ~ 1-100 GOPS ~ 0.01-0.1 s latency
Missile Seeker Stream Video Processing

256x256 Focal Plane Array @ 60 FPS

2000

Processing Block | Opcount (MFLOPS)
---|---
NUC | 65 — 2000
Dither Subtraction | 15 — 1800
Dempster-Shafer Plausibility Computation | 4 — 100
Total | 310 — 7300+

Latency ~ 1mS
Outline

- DoD High-Performance DSP Applications
- Middleware (with some streaming constructs)
- Some Key Features
- Future Directions
- Summary
**Mappably Stream Processing**

**Decomposable**
into Tasks (computations) and Conduits (communications)

**Mappably**
to different sets of hardware

**Measurable**
resource usage of each mapping

- Each compute stage can be mapped to different sets of hardware and timed

- Beamform: $X_{OUT} = w \cdot X_{IN}$
- Filter: $X_{OUT} = \text{FIR}(X_{IN})$
- Detect: $X_{OUT} = |X_{IN}| > c$
Parallel Vector Library Layered Architecture

Application

Parallel Vector Library

Hardware

Input

Stream Processing

Output

Matrix S, vector s stream

F(s)

Conduit

Task

Grid

Map

Distribution

Math Primitives

Communication Primitives

Library “API”

Library “ISA”

Workstation

Intel Cluster

PowerPC Cluster

Embedded Board

Embedded Multi-computer

MIT Lincoln Laboratory
### Parallel Vector Library Components

<table>
<thead>
<tr>
<th>Stream Processing &amp; Control</th>
<th>Description</th>
<th>Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Class</strong></td>
<td><strong>Description</strong></td>
<td><strong>Parallelism</strong></td>
</tr>
<tr>
<td>Stream Vector/Matrix</td>
<td>Streams organized as matrices or vectors that span multiple processors</td>
<td>Data</td>
</tr>
<tr>
<td>Function</td>
<td>Performs signal/image processing functions on matrices/vectors (e.g. FFT, FIR, QR)</td>
<td>Data &amp; Task</td>
</tr>
<tr>
<td>Task</td>
<td>Supports algorithm task decomposition (the nodes in a signal flow diagram)</td>
<td>Task &amp; Pipeline</td>
</tr>
<tr>
<td>Conduit</td>
<td>Supports stream movement between tasks (interconnection in stream flow graph)</td>
<td>Task &amp; Pipeline</td>
</tr>
<tr>
<td>Mapping</td>
<td><strong>Description</strong></td>
<td><strong>Parallelism</strong></td>
</tr>
<tr>
<td>Map</td>
<td>Specifies how Tasks, Streams, and functions are distributed over processors</td>
<td>Data, Task &amp; Pipeline</td>
</tr>
<tr>
<td>Grid</td>
<td>Organizes processors into a 2D layout (virtual machine model) - Hierarchical</td>
<td></td>
</tr>
</tbody>
</table>
Some Key Features

• The application stream flow is defined by *tasks* interconnected by *conduits*.
  – Defines an application macro-architecture
    Presents opportunities for optimization at construction time
  – Tasks can contain other tasks and conduits
    Hierarchy helps to handle complex applications
  – Conduits can reorganize data and handle buffering
    Corner turns, multi-rates, different granularities

• *Streams* flow through conduits and are transformed in tasks by *functions*.
  – Streams are resizable with fundamental epochs defined by vector or matrix dimensions. They can be produced and consumed piecemeal.
  – Function implementations specialize based on stream arguments (and maps.)

• All objects are *mappable* and hence implicitly *parallel*.
  – Functionality is map invariant
  – Scalable and portable (virtual machines)

• Re-mapping gives support for fault recovery and for automated performance tuning.
Outline

- DoD High-Performance DSP Applications
- Middleware (stream language prototyping)
- Future Directions
- Summary
Future Directions

- Standardization of parallel signal/image processing middleware:
  - HPEC-Software Initiative (VSIPL++)
- Generative programming techniques
  - Efficient implementation of multi-expression program blocks
  - Breaking the “abstraction barrier” without using streams
- Extensions of middleware to heterogeneous platforms
  - FPGAs, DSPs, Microprocessors
- Run-time optimal mapping
  - Managing latency, throughput, power,...
Generative Programming Approach:
Expression Templates + Comma Operator

Problem: “Abstraction Boundary”
- Related expressions: Vector/Matrix expressions sharing common variables
  \[ \text{OUT} = \text{IN} > \text{THRESH} \]
  \[ \text{THRESH} = 0.9 \times \text{THRESH} + 0.1 \times \text{IN} \]
- Efficient implementation requires that data used in more than one expression remain in register/cache
- How to do this and maintain high-level API?
  - Hand-coded “for” loop works, but is too low level
  - Expression templates (PETE) can build a compilable expression for a single statement, but cannot span multiple statements

Solution: PETE combined with Comma Operator
- Approach: Use comma operator to build expression list
- PETE Expression list destructor triggers evaluation when semicolon is reached
  \[ \text{out} = \text{in} > \text{thresh}, \]
  \[ \text{thresh} \&= 0.9 \times \text{thresh} + 0.1 \times \text{in}; \]
Optimal Mapping of Stream Algorithms

Example - Find:
Min(latency | #CPU)
Max(throughput | #CPU)

Example - Use:
Genetic Algorithm
Decision Directed Learning
Predicted and Achieved Latency and Throughput

Problem Size

Small (48x4K)  Large (48x128K)

- Graph Solution selects correct optimal mapping
- Good agreement between predicted and achieved latencies and throughputs
Summary

• High-performance stream processing for DoD applications requires 100’s GOPS in stringent form factors.
  – Programmable solutions are parallel processors
  – Computational efficiency is paramount

• Middleware approaches emerging today have attributes of stream languages except:
  – Lack rigorous formal definitions (and are incomplete)
  – Do not have good compiler support (efficiency issues)
  – BUT: they provide high productivity compared to traditional approaches AND better lifecycle support

• MIT Lincoln Laboratory is in the business of evaluating new technologies for DoD signal and image processing applications
  – Middleware, languages,…
  – FPGAs, VLSI, DSP, PCA, …
PETE Review

Step 1: Operators Form expression

\[ A = B + C \times D \]

BinaryNode<OpAdd, float*, Binary Node<OpMul, float*, float*>>

Step 2: ‘=’ operator evaluates expression

**Vector Operator**: 
```
int it = begin();
for (int i=0; i<size(); i++) {
    *it = forEach(expr, DereferenceLeaf(), OpCombine());
    forEach (expr, IncrementLeaf(), NullCombine());
    it++;
}
```

**Action performed at leaves**

**Action performed at internal nodes**

PETE ForEach: Recursive descent traversal of expression
- User defines action performed at leaves
- User defines action performed at internal nodes

**User specifies what to store at the leaves**

**Translation at compile time**
- Template specialization
- Inlining
PETE and the Comma Operator

Step 1: Operators form expression list

out = in > thresh,
thresh = 0.9*thresh + 0.1*in;

Step 2: Expression list destructor triggers expression evaluation

Expression list destructor
if (isBaseNode)
{
  for (int i=0; i<size; i++)
  {
    forEach(expr, DereferenceLeaf(), OpCombine());
    forEach(expr, IncrementLeaf(), NullCombine());
  }
}

Comma orders evaluation from left to right

*out = in1 > *thresh1;
*thresh2 = 0.9 * *thresh3 + 0.1 * *in2;
out++; in1++; thresh1++;
thresh2++; thresh3++; in2++;

PETE ForEach changes:
- ‘=’ operator behavior with OpCombine: assign from RHS to LHS
- ‘,’ operator behavior: first evaluate left expression, then evaluate right expression
Scalable Approach

```c
#include <Vector.h>
#include <AddPvl.h>

void addVectors(aMap, bMap, cMap) {
    Vector< Complex< Float> > a('a', aMap, LENGTH);
    Vector< Complex< Float> > b('b', bMap, LENGTH);
    Vector< Complex< Float> > c('c', cMap, LENGTH);

    b = 1;
    c = 2;
    a = b + c;
}
```

- Single processor and multi-processor code are the same
- Maps can be changed without changing software
- High level code is compact
Corner Turn Operation

All-to-all communication

Half the data moves across the bisection of the machine

Filter \rightarrow Beamform

Original Data Matrix → Corner-turned Data Matrix
All PVL objects contain maps

PVL Maps contain:
- Grid
- List of nodes
- Distribution
- Overlap

List of Nodes: \{0,2,4,6,8,10\}

Map

Vector/Matrix Stream

Function

Conduit

Task

Grid

Distribution

Overlap