User Case Study:
ISI programming with RAW

Matt French (mfrench@isi.edu), Steve Crago (crago@isi.edu)
Application Mappers: Lakshmi Srinivasan, Lavanya Swetharanyan, Gunjan Dang
Routing & Scheduling: Dong-In Kang (dkang@isi.edu), Sumit Lohani
Hardware & Firmware: Li Wang (lwang@isi.edu), Chen Chen (chen@isi.edu)
March 6th, 2003
How we use the system

- Starsearch (RAW User code)
  - All user code checked into cvs repository hosted by MIT
  - ISI code at: starsearch/end-to-end/isi

- RAW Tools
  - Cagfarm (MIT)
    - Latest toolset
    - Local resources limited
    - Batch of long simulations
  - Local (ISI)
    - Editing
    - Shorter length simulations (< 1 hr)
    - Network reliability (cagfarm maintenance?)
  - Our perspective: Functionality and RAW performance identical
First Steps

- Basic examples starsearch/examples
  - include files
  - Makefile
- Starsearch is an example!
- Biggest help:
  - Starsearch mailing list archives
  - Raw cross reference at: http://www.cag.lcs.mit.edu/lxr/
8x8 Tile Wideband GMTI

- Proof of concept for ISI routing tool chain
  - complex routing patterns not seen in 4x4
- Largest tile size we can reasonably simulate
  - ~1 day per simulation

Parameters
- PRF 1,000 Hz
- Transmit duty factor 10%
- Sampling Frequency 500 KHz
- Channels 8
- PRIs 48
- PRI staggers 2
- Subbands 2
- Post-ABF beams 4
- Post-STAP beams 2
- Range Gates 450
- Range Gates in Subband 225
- Subband Analysis Filter Taps 24
- Subband Synthesis Filter Taps 32
- Time Delay & Equalization Filter Taps 32
- Pulse Compression Filter Taps 16
- Dopplers out of Doppler Filter 47
- CPI Length 0.048 s
Communication Programming Model

- Communication
  - Static
    - Implicit
      - StreaMIT
  - Explicit
    - Programmer Optimized
  - Dynamic
    - Simple (Non-buffered)
      - Static Communication API
    - rMPI
Explicit Tile Partitioning Assumptions

- Problem: find minimum # of tiles to meet real-time constraints
- Computation: \#Tiles = (Flops/s)/(P*STU*MTU)
  - P - Processor Speed - 250 MHz
  - STU - Single Tile Utilization - Flops/cycle - 50%
  - MTU - Multi-Tile Utilization - Percent of single tile utilization achieved when multiple tiles are connected - 50%
- Load balance

- Memory
  - Size to fit local data memory - 32KB
  - Cache misses use dynamic network

- I/O
  - Included in Flops/s calculation
  - Leverage Static network - 1 cycle throughput
  - Most applications: Computation Time >> Communication Time
8x8 GMTI Tile Designation

Subband, TDE
No of tiles = 32

ABF
No of Tiles = 4

Doppler Filtering, Pulse Compression
No of tiles = 16

STAP
No of Tiles = 6

Subband Synthesis
No of Tiles = 4

Detection, Parameter Estimation
No of Tiles = 2
Computation: Subband Analysis

- Map Matlab to C/ASM
  - Expose parallelism
  - Data dimensionality
  - Categorize computation
    - Initialization
      - pre-compute and store
      - Filter taps, FFT weights
    - Real-time Stream
      - FFT, Complex Multiply
- Divide larger loops over tiles
  - # Range Gate Vectors = Nch * Npri = 8*48 = 384
  - Vectors / Tile = 384 / 32 = 12
  - Load balanced!

Matlab

```matlab
%filter taps will be calculated once, can store in memory
% in C versions use static values listed in subband_filter.dat

%perform filter operation, FFT - mult - IFFT
for subband = 1:nsub,
    i_filter_data(subband,:) = fft(demod_data(subband,:), fft_leng) .* F_filter_taps; %F_filter_taps is pre-computed static filter Freq response
    filter_data(subband,:) = ifft(i_filter_data(subband,:),fft_leng);
end
```
Virtualized Input stream by writing bC code
- read from file
- write to static network North port
- also have output equivalent

Top row passes data from North port to lower tiles

Could hand write switch code
Output of one Subband row communicates to one Adaptive Beamforming Tile

Scheduling and code automation becomes a necessity
Communication Lessons

- Control Switch Tightly
  - Use BNEZD, BEQZD to hop from one communication pattern to another vs interrupt from tile processor

- Deadlock Avoidance
  - Local tile needs global routing to unroll communication

- Stalls can propagate
Current ISI non-API Tool Flow

Tile Processor

- routes.info
- codegen.pl
  - routes_tile.h
  - main.c

Route Pair Creation Tool - routesgen.c

Routing Path Optimization Tool - routingIncr
  - routes.input
  - tilegen.pl
  - routes.rt

Switch code generator-switchgen.pl
  - routes.tr
  - routes.S
  - routes.h
  - route_lib.c

rgcc
  - .elf

Switch Processor
Routesgen

- C code definition
- Switch
  - Source destination pairing
  - Communication Length
- Tile Processor
- Output / Input buffers

**route.input**

**route.info**

Src-dest pairs, src addr, dest addr, pkt len

Task0_0_receive_data_0
36 - 51, &abf_out[0], &doppler_in[0], 26
37 - 51, &abf_out[0], &doppler_in[26], 26
38 - 51, &abf_out[0], &doppler_in[52], 26
39 - 51, &abf_out[0], &doppler_in[78], 18
Routing Path Optimizer

- Packs routes together for Optimality
- Breaks communication into sub-stages where necessary

Src-dest pairs subdivided into stages

Task0_0_receive_data_0_1
36-35-43-51
Task0_0_receive_data_0_2
37-36-35-43-51
Task0_0_receive_data_0_3
38-37-36-35-43-51
Task0_0_receive_data_0_4
39-38-37-36-35-43-51
Tilegen

- Count number of hops to each switch
- Delay first execution of a path by difference of number of hops to switch

<table>
<thead>
<tr>
<th>Tile #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prolog</td>
<td>P-&gt;S</td>
<td>P-&gt;S</td>
<td>N-&gt;E</td>
<td>N-&gt;W</td>
</tr>
<tr>
<td>Body</td>
<td>P-&gt;S</td>
<td>P-&gt;S</td>
<td>N-&gt;E; E-&gt;P</td>
<td>N-&gt;W; W-&gt;P</td>
</tr>
<tr>
<td>Epilog</td>
<td></td>
<td></td>
<td>E-&gt;P</td>
<td>W-&gt;P</td>
</tr>
</tbody>
</table>

Switch configuration and hop count

*Task0_0_receive_data_0*

- Tile35 1:e-s
- Tile36 0:p-w
- Tile43 2:n-s
- Tile51 3:n-p
Switchgen

- Switch code generation
- Library of possible calls
- Switch code includes prologue / epilogue delays

```c
void setup_Task0_0_receive_data_0_1_35();
void send_Task0_0_receive_data_0_1_35(int length, int *data);
void recv_Task0_0_receive_data_0_1_35(int length, int *data);
void send_stride_Task0_0_receive_data_0_1_35(int length, int *data, int stride);
void recv_stride_Task0_0_receive_data_0_1_35(int length, int *data, int stride);
void blocking_pass_Task0_0_receive_data_0_1_35(int length);
void nbblocking_pass_Task0_0_receive_data_0_1_35(int length);
void end_nbblocking_pass_Task0_0_receive_data_0_1_35();
```

route.h

C code for all the functions in route.h

route_lib.c

Assembly code for the switch functions
Tile processor code generation

Accounts for auto-generated sub-stages

Source Code for all the tiles for different Communication rounds

```c
#define Task0_0_35 {
    blocking_pass_Task0_0_receive_data_0_1_35(26);
    blocking_pass_Task0_0_receive_data_0_2_35(26);
    blocking_pass_Task0_0_receive_data_0_3_35(26);
    blocking_pass_Task0_0_receive_data_0_4_35(18);
}
#define comm_0(COMM_RND, TILE#) {
    switch (COMM_RND) {
        case :
            switch(TILE#) {
                case 35: Task0_0_35;
                .......
            }
    }
}
Static Communication API Development

Source code

Development w/ MPI
Augment w/ Routing APIs

Compile & Run

Static routing Information

Routing Tool

Final Routing Input
Routing Table in a C file

Code Generation Tool

Switch Setup Code

Compile

Run on RAW
Static Communication API Tools

- Profiling
  - Generate communication pairs
  - Allow scheduling flexibility while meeting constraints
- Routing
  - Maximize throughput (avoid congestion)
  - Avoid deadlocks
- Code generation
  - Switch code (hidden from user)
  - Switch set-up code
```
#include <route.h>
main() {
    route_struct route_result;         int * receive_buffer;
    init_route(my_tile_id); // initialize routing table
    // do computation if needed
    for(i = 0; i < N ; i++) {
        // do computation if needed
        route_setup( stage_id, comm_id, OP_RECV, my_tile_id, src_tile_id, dest_tile_id,
        amount_of_comm, & route_result); // set up switch

        receive_buffer = user_function(route_result); // check which route
        // do computation if needed
        #ifdef ROUTE_MODE
            MPI_RECV(receive_buffer, route_result->len  );
        #else
            receive(receive_buffer, route_result->len);
        #endif
    }
    #ifdef ROUTE_MODE
        MPI_Finalize();
    #else
        close_route(my_tile_id, ROUTE_MODE);
    #endif
}
```
Static Communication API Status

- Tool flow functional
- Applications/modules
  - Corner turn
  - GMTI sub-band analysis
  - Multi-tile FFT
- Performance
  - <20% overhead for multi-tile FFT
  - Overhead decreases as bigger data chunks are exchanged
Debugging

- Debug / Develop Computational Kernel on x86 first
- Debug Raw communications w/o computation
  - Make sure no deadlock
  - Use data test patterns to tag if correct data goes through correct path
- Integrate Computation and Communication
  - Verify results against x86 output
- Typically use gmake run with raw_test_pass or printfs
- Nothing beats btl simulator
## Initial Subband Results

<table>
<thead>
<tr>
<th>Function</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modulation per vector</td>
<td>17,413</td>
</tr>
<tr>
<td>FFT per vector</td>
<td>126,660</td>
</tr>
<tr>
<td>Freq Domain Multiply per vector</td>
<td>19,205</td>
</tr>
<tr>
<td>IFFT per vector</td>
<td>147,482</td>
</tr>
<tr>
<td>Downsampling per vector</td>
<td>4,731</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td><strong>315,491</strong></td>
</tr>
<tr>
<td>Total Subband (12 PRI vectors, 2 subbands)</td>
<td>7,571,784</td>
</tr>
<tr>
<td>TDE (FFT-MULT-IFFT) Total</td>
<td>3,157,272</td>
</tr>
<tr>
<td>Communication</td>
<td>1,138,224</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>11,867,280</strong></td>
</tr>
<tr>
<td>Time @ 250 Mhz</td>
<td>0.0474 s (CPI 0.048 s)</td>
</tr>
</tbody>
</table>
Applications
- Linking more stages together
- Update FFT - DITF w/ loop indexing yields ~2x
- Develop a 4x4 Narrowband GMTI demo

Static API
- Cleaning up calls
- Integrating improvements

Hardware
- Bringing up Raw chip @ ISI
- Improving rawdb to Raw chip bandwidth