Deriving FPGA Based Application Specific Architectures for Mission Planning Algorithms

Aravind Dasu and Jonathan Phillips

Utah State University

UtahState UNIVERSITY

Outline

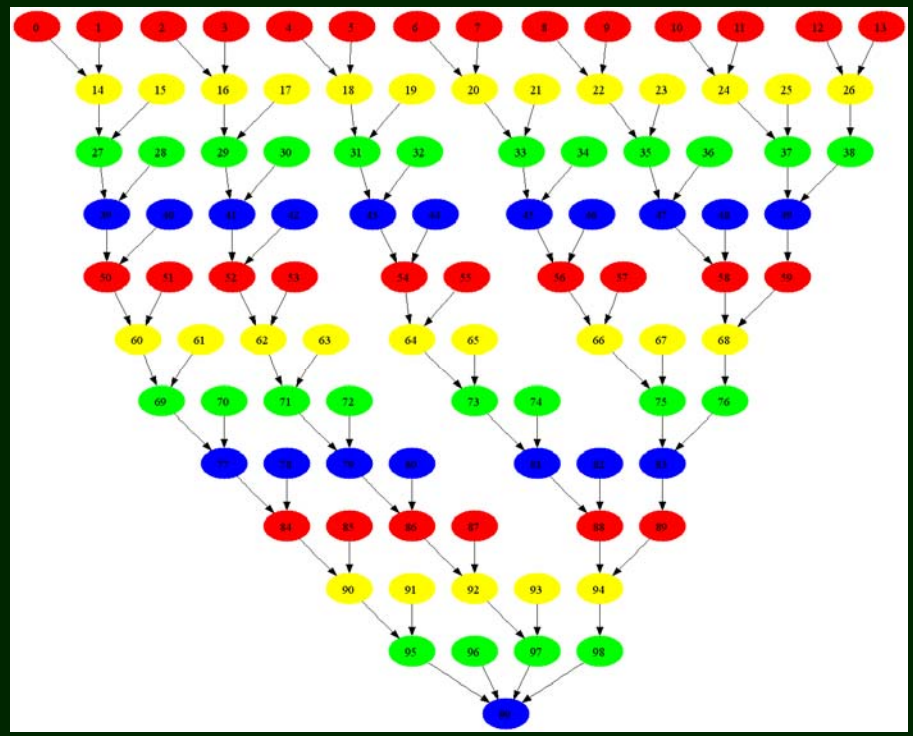
- Introduction and Background
- Architecture Template
- Results
- Conclusions and Future Work

Relevance to NASA's missions

- Deep space exploration missions cannot rely on constant and reliable communication between earth and spacecraft
- **Necessitates greater autonomy through advanced on-board managed command and control software and hardware systems**
- Vital efforts from NASA on the software side
  - ASPEN and CASPER
    - JPL
    - Automated scheduling/activity planning software
    - Use iterative repair/simulated annealing type algorithms

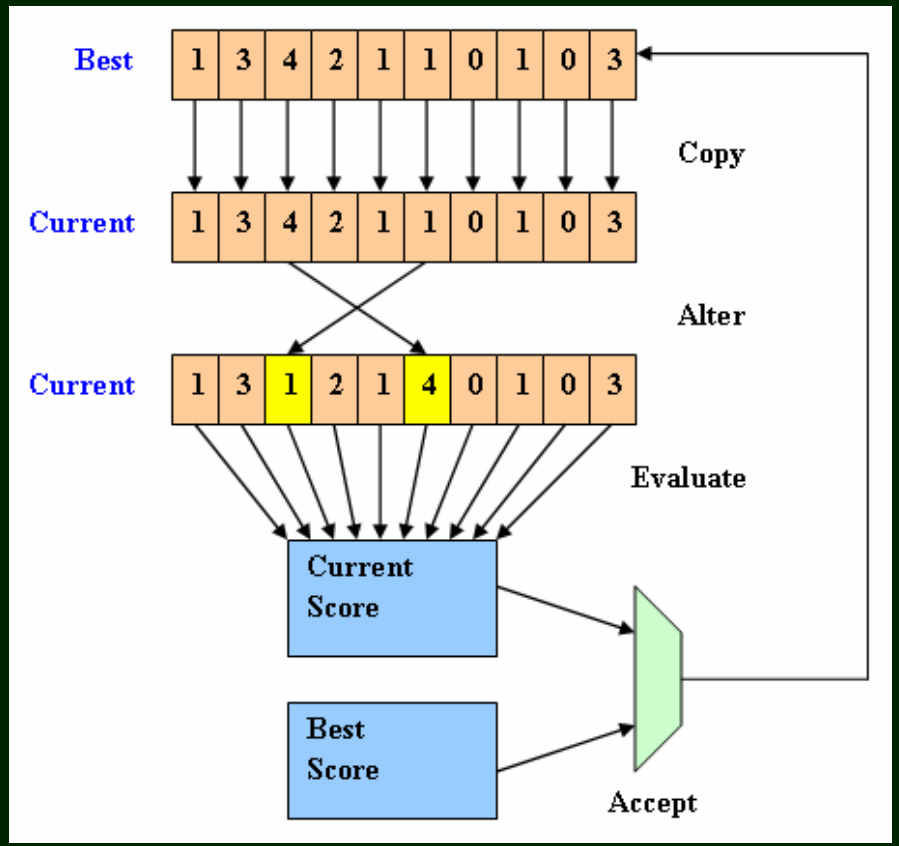
Event Scheduling

- Given
  - a set of events
  - a set of dependencies between events
  - a set of resources used by events
- Find an efficient schedule for all events
- NP-hard problem



Iterative Repair and Simulated Annealing

- Solution consists of an array of start times
- Initial solution is gradually modified over many iterations
- Simulated Annealing prevents entrapment in local optima



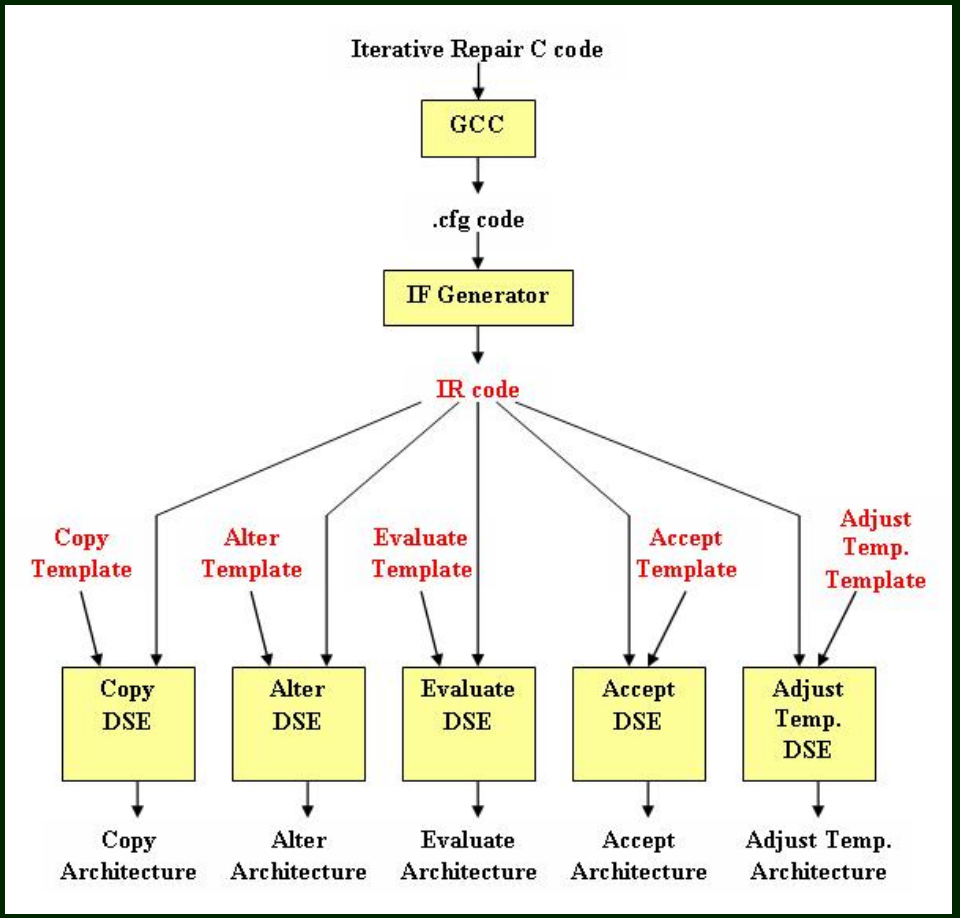
On Board Processing Options

- General-purpose processors (PowerPC, MIPS, ARM, x86, etc.)
  - *Software programmable/reusable*
  - *Not optimized for any application – baseline performance*
  - *Workhorse of today's space computers*
  - *Emerging applications will be too complex for these devices*
- ASICs:
  - *Chip can be customized for application*
  - *The best you can do for an application*
  - *Can result in orders-of-magnitude speedup over conventional processors*
  - *Too expensive for low volume market*

On Board Processing Options

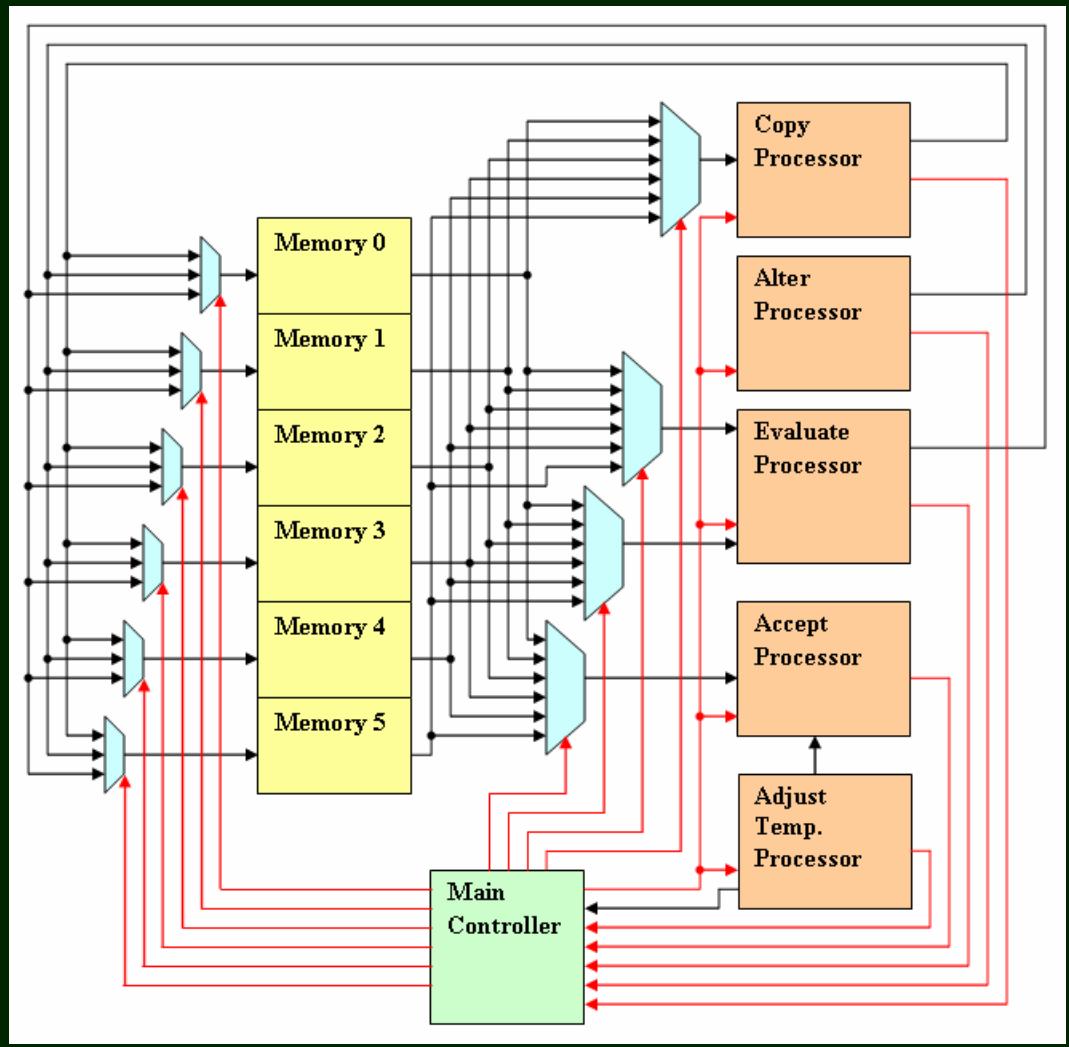
- FPGAs
  - Configurable/reconfigurable
  - Already in use in space missions
    - MARTE
    - New Millenium 8
    - New Frontier
- Encouraging and Emerging companion technologies
  - Dependable multiprocessor system
    - NASA sponsored: Honeywell, U. Florida
    - Multi-FPGA system
  - Carbon Nano Tube based FPGAs
    - NRO + NASA effort
  - Cumbersome to use/program if user is not a VLSI architecture expert

Design Flow



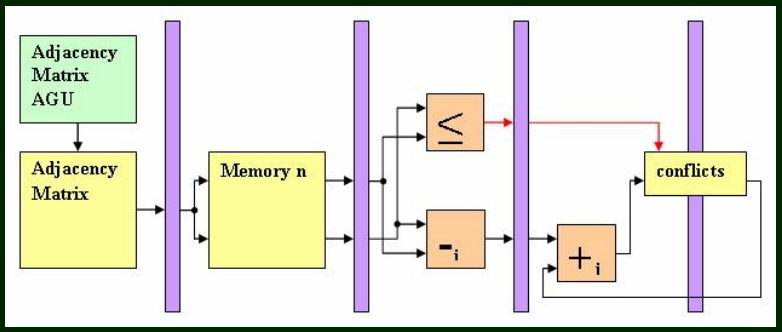
application-specific architecture for Iterative Repair

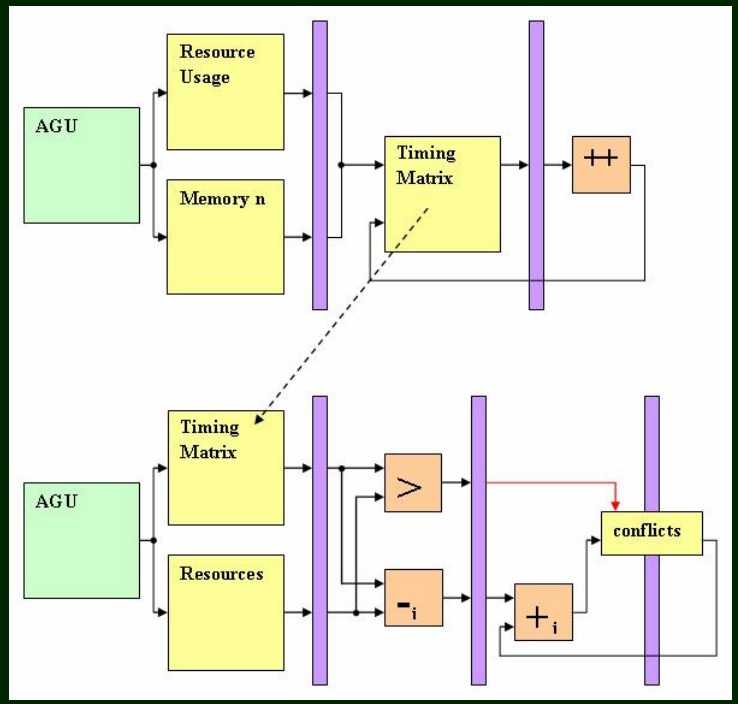
Architecture Template Diagram

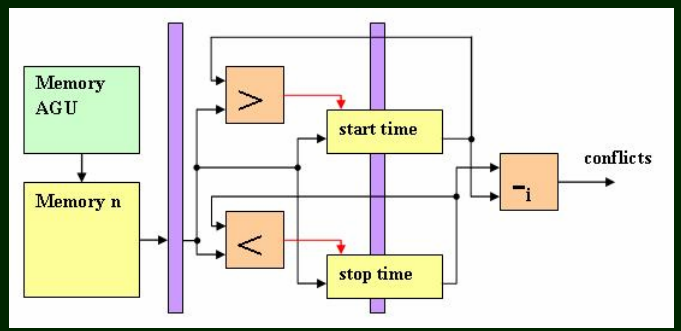


Evaluate Processor

- Computes solution penalty based upon
  - Dependency graph conflicts
  - Total schedule length
  - Resource over-utilization
- Parallelism

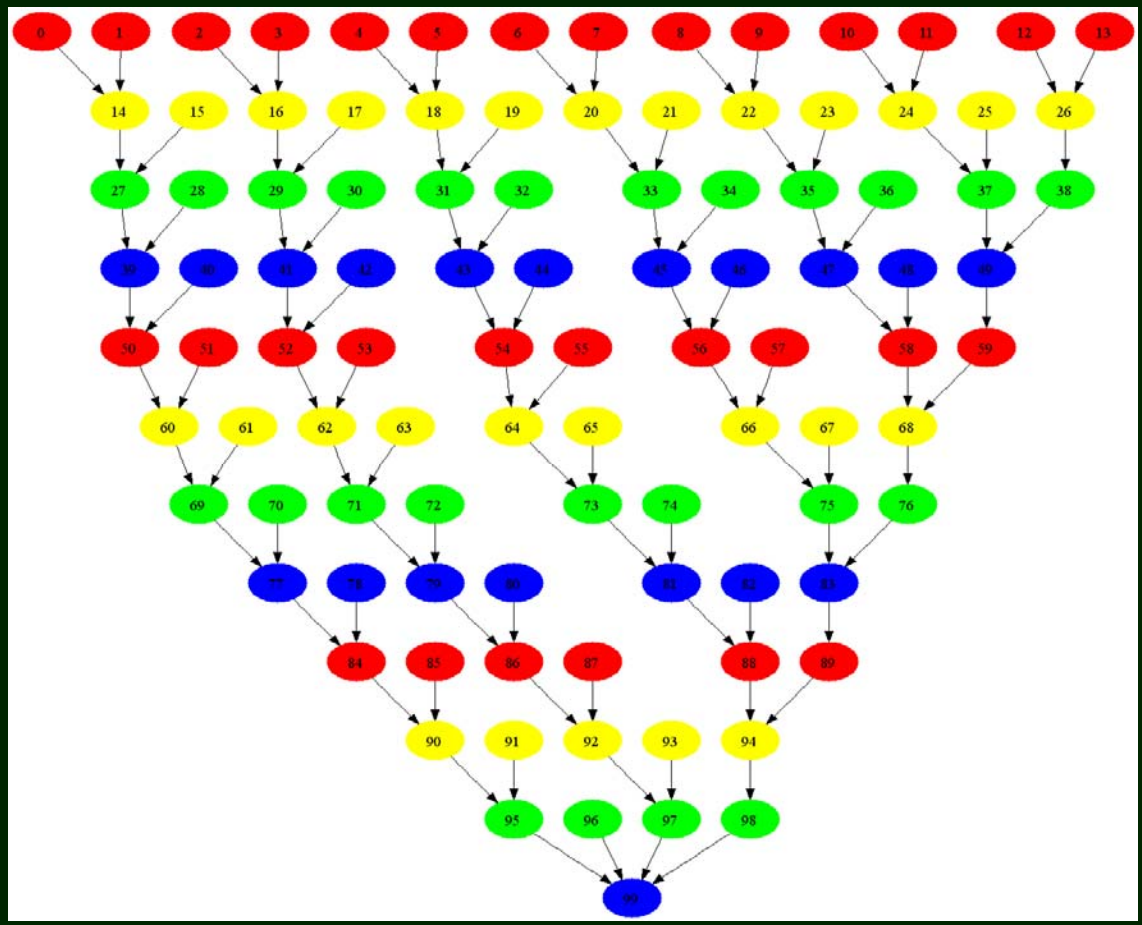






Example

- Given
  - 100 tasks
  - 4 resource types (red, yellow, green, and blue)
  - 4 units of each type of resource
  - Maximum latency of 32 cycles
- Find optimal schedule



Results for prototype architecture

	Slice Count	DSP48 Units	BRAMs	Latency	Max. Freq. (MHz)
Main Controller	193	0	0	3	472
Copy Processor	21	0	0	101	307
Alter Processor	390	0	0	21	238
Eval. Processor	393	0	5	235	310
DGVP	104	0	1	102	347
TSLP	74	0	0	101	352
ROP	215	0	4	233	310
Accept Processor	1,424	0	1	54	197
Adjust Temperature Processor	186	4	0	12	445
Memory Module	1,612	0	24	N/A	136
Complete Processor	**4,712**	**4**	**35**	**235**	**136**

Results for prototype architecture

Processing Platform	Clock Frequency	Cycles	Wall Clock Time	Speedup
Xilinx Virtex-4 embedded PowerPC core	100 MHz	1.87×10^{10}	187.1 s	N/A
AMD Athlon 64	2.61 GHz	3.7×10^{10}	14.265 s	13.11
Xilinx Virtex-4 Iterative Repair circuit	136 MHz	184,198	318 ms	**588.4**

Conclusions

Why is the custom approach so much faster?

- 5-stage pipeline allows multiple solutions to be in process
- Most complex stage (Evaluate) is parallelized to minimize latency
- Application specific architecture removes wasted load/store cycles that are common in general purpose processors

Future Work

- Templates are complete – must now be integrated into tool flow
- Testing
- Design Space Exploration
- Architecture to VHDL converter
- Fault mitigation/Tolerance