HPEC 2004

Gedae: Auto Coding to a Virtual Machine

Authors: William I. Lundgren, Kerry B. Barnes, James W. Steed
What is Gedae?

Gedae is a block diagram language...

Express signal and data processing algorithms, parallelism, load balancing, fault tolerance and mode control.
..that Gedae **transforms** under user control...

User can set optimization parameters that are independent of the graph to guide transformation
...to operate efficiently on a virtual machine.

Complete systems can be developed independent of the target system without losing runtime efficiency.
Gedae Language

• Gedae provides application information through
  – modules with well-defined behavior
  – ports with well-defined characteristics
  – and manifest connectivity with explicit sequential and parallel execution paths
• This information is implicit in most languages
• Gedae makes the information explicit
  – over 50 different information expression features

Information provided by language allows Gedae to analyze and efficiently implement algorithms
The Gedae transformations build a detailed model of the deployed application. Gedae uses that information to provide visibility
Gedae Virtual Machine (VM)

- Gedae provides the following components:
  - Command handler
  - Dynamic scheduler
  - Segmentation Support
  - Primitive Support
  - Visibility Support
- The vendor provides
  - Inter-processor communications
  - Optimized vector libraries
  - Other basic services

The Gedae virtual machine makes applications processor independent
Three Examples

- Real-Time Space-Time Adaptive Processing (RT-STAP)
  - Miter benchmark graph
  - Illustrates efficient parallel execution of large graph
- Multilevel Mode Graph
  - Illustrates nested mode control with distributed state
  - Dynamic data application
- Sonar Graph
  - Illustrates large data reduction during processing

Each example illustrates features of the language, transformations, and virtual machine
Families permit replicating box and data elements
RT-STAP: Language

- Instantiation constants control the size of the graph
- Routing boxes allow equation based connectivity

Gedae, Inc.
www.gedae.com
**RT-STAP: Transformations**

- User maps primitives to physical processors
- Gedae transforms graph by inserting send/receive primitives to communicate between partitions
- Gedae automatically creates executables to run on each processor

Different mappings can be tried without modifying the graph – the needed transformation happens automatically.
RT-STAP Transformations

- User can set transfer properties on send/recv pairs with Transfer Table
- Transformations automatically set parameters to send/recv pairs to communicate these properties to running application

User can guide transformations to optimize implementation

Gedae, Inc.
www.gedae.com
RT-STAP: Running on VM

Send/Recv webs show interprocessor communication and uncover synchronization problems

Gedae, Inc.
www.gedae.com
RT-STAP: Running on VM

Preplanned use of memory allows distributed runtime debugging

Gedae, Inc.
www.gedae.com
Mode Control: Language

- Branch boxes make mode changes and mark segment boundaries
- “Exclusive” branch outputs show where resources can be shared
- State shared between modes is explicitly declared in the graph

The Gedae primitive language directly supports segmented data processing, sharing of resources, and distribution of state.
Mode Control: Language

Branch box copies input data stream to one of a family of outputs based on a control stream. Output is:

- **Segmented** - the box will add segment boundaries to the output
- **Dynamic** - the box will state how much data is produced on the output at runtime.
- **Exclusive** - only one of the family of F outputs gets data on any firing. Allows sharing of resources and state.

The Gedae extensible language has no “built-in” primitives. 8000+ delivered primitives. Users can add custom primitives.

```c
Name: cp_branchf_e
Input: stream ControlParamRec in;
Input: stream int c;
Local: int last;
Output: exclusive segmented dynamic stream
        ControlParamRec [F]out;
Reset: { last = -1; }
Apply: {
    int g,i;
    int prdc = 0;
    for (g=0; g<granularity; g++) {
        int j = c[g];
        if (last != j) {
            if (0<=last && last<F) {
                produce(out[last],prdc);
                prdc = 0;
                segment(out[last],SEGMENT_END);
            }
            last = j;
        }
        if (0<=j && j<F) {
            *out++ = *in;
            prdc++;
        }
        in++;
    }
    produce(out[last],prdc);
}
```
User can set partitioning, mapping, data transfer methods, granularity, priority, queue sizes and schedule properties from the group control dialog

www.gedae.com
Mode Graph: Running on VM

- Each mode requires a different number of processors
- Branch boxes at one level are responsible for the dynamic distribution

VM runtime kernel enforces dynamic data driven execution. Send and receive primitives and state transfer primitives use BSP of virtual machine to transfer data.
Mode Graph: Running on VM

• Primitives to send and receive state are automatically added by transformations

• Messages generated by Virtual Machine at mode change boundaries efficiently coordinate state transfers

Result is efficient transparent use of shared state on distributed processing system
Sonar Graph creates low bandwidth output from high bandwidth input data
Sonar: Language

- Connectivity + Port Descriptions gives information needed to schedule graph.
- `mx_vx` produces $R=120$ tokens out for every 1 token in.
- `vx_multV` box must fire 120 times for each firing of the `mx_vx` box.
- `vx_fft` box fires one time for each firing of `vx_multV` box.
- Simple predetermined schedule generated from graph and info embedded in primitives.

Can create a multirate graph that has boxes firing at different granularities.
Sonar: Transformation

- User can place boxes in subschedules to strip-mine the vector processing
- Allows use of fast memory
- Can reduce memory usage

Multirate graphs can be implemented using subscheduling to improve speed and reduce memory usage
Sonar: Transformation

Auto-Subscheduling Tool

• User can put boxes into named subschedules manually – but can be difficult
• Auto-Subscheduling Tool puts boxes in subschedules automatically
• Finds nested sets of connected boxes running at common granularities.
• Automatically sets subscheduling levels

Auto-subscheduling has reduced memory needed by graph from 250 Mbytes to about 2.5 Mbytes - 100x improvement

Gedae, Inc.
www.gedae.com
Sonar: Running on VM

Multiple levels of subscheduling evident on Trace Table

Gedae, Inc.
www.gedae.com
Conclusion

- Gedae Block Diagram Language allows simple expression of a wide range of algorithms
- User optimization information can be added without modifying block diagram
- 100+ transformations create efficient executable application from language and user information
- Application runs efficiently on Virtual Machine
- VM provides portability and visibility