Gedae is an integrated application development environment. It has been under development since 1987 – though the concepts involved are rooted in much earlier work done in the areas of data flow and hardware simulation. In Gedae we have developed a language for describing an architecture-independent functional specification, a virtual machine on which the application runs, and transformations that create an efficient implementation of the application that runs on the virtual machine. In this paper we discuss three topics – the language, the virtual machine and the transformations.

The language was developed with two requirements – any functionality must be easily expressible, and the language must be transformable into an efficient implementation on the virtual machine. The Gedae Language consists of both the Gedae Primitive Language and the Gedae Graph Language. Much of the expressiveness is in the primitive description language. The language has over 50 expression features to define the behavior of functional ports. Port data flow requirements can be specified either prior to runtime (static) or at runtime (dynamic). Ports can add segment boundary markers on the data flow streams, thereby breaking the stream into independent data sets. Exclusive families of ports can send data down one branch or another to implement mode changes while maintaining coherent state vectors used by all the modes. Primitives can maintain their own local state variables and provide methods for execution, startup, termination and handling the beginning and ending of segment processing. The Gedae Graph Language allows the hierarchical development of graphs consisting of primitives, parameters and other Gedae graphs. The graph language can describe families of these entities to allow parameterized expression of parallelism. The resulting language permits direct expression of signal and data processing algorithms, distribution for providing load balancing and fault tolerance, and application (or software, or mode) control.

To achieve efficiency, the language and virtual machine were codesigned. The virtual machine contains a runtime kernel that executes components generated by the transformations. For example, the static scheduler executes predetermined execution sequences based on static data flow ports, and the dynamic scheduler executes groups of static schedules that interface through dynamic data flow ports. The virtual machine manages the segment processing and controls the efficient and timely transfer of distributed state vectors between processors. The virtual machine also allows for vendor specific optimizations of processing, such as, setting data transfer parameters. A thin layer
Gedae: Auto Coding to a Virtual Machine

Approved for public release, distribution unlimited

See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.
over the vendor-provide vector processing libraries allows primitives to execute efficiently.

One of the unique features of Gedae is the visibility of the implementation and the execution it provides. This visibility is possible because the language, the transformations and the virtual machine are all part of Gedae. The visibility allows the generation of detailed execution timelines and the symbolic viewing of any memory in the system. Primitive execution, queue state and data transfers between processors can be dynamically viewed when the application is running.

The transformations are the central part of Gedae and make possible the efficient execution of the application expressed in the Gedae Language on the Gedae Virtual Machine. The transformations are fully automated but can be guided by user supplied implementation parameters to control distribution, strip mining, data transfers, scheduling priorities (both static and dynamic), queue policies and memory management. Some of the transformations directly modify the graph into an equivalent graph to implement a user entered implementation decision. For the user to distribute a graph, the user specifies a partitioning of the graph and a mapping of the graph to individual processors. Gedae modifies the graph by inserting send and receive primitives that run on the separate processors and maintain the data flow and connectivity of the graph. The user does not have to modify the graph to achieve these results.

For example, the following graph has dynamic queues and is distributed to four processors by the user:

It is transformed into a new graph, as seen below, with send and receive boxes inserted to manage communications and dynamic queues also inserted to handle dynamic data flow boundaries. Other transformations include modifying the graph to implement strip mining of vectors, adding primitives to implement delay, and adding primitives to allow communication of the graph to the host program or to other Gedae applications. Data structures are also created to implement segmentation, mode control and distributed state coherency.

A sonar signal processing graph will be used to demonstrate how a graph is transformed into an implementation. It will be shown how the transformations can be used to modify the graph execution without changing the Gedae Language expression of the graph. The resulting implementations will be contrasted with how the same implementations would be achieved using traditional programming techniques.
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

Gedae, Inc.
www.gedae.com
...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
Gedae Transformations

- The block diagram is transformed using over 100 algorithms.
- The transformations establish the:
  - Order of execution
  - Queue sizes
  - Granularities
  - Memory layout
  - Dynamic schedule parameters
  - Data transfer types and parameters
  - Mode control

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

Gedae, Inc.
www.gedae.com
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
RT-STOP: Language

Families permit replicating box and data elements

Gedae, Inc.
www.gedae.com
RT-STAP: Language

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

Gedae, Inc.
www.gedae.com
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
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.

```plaintext
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);
}
```
Mode Graph: Transformation

User can set partitioning, mapping, data transfer methods, granularity, priority, queue sizes and schedule properties from the group control dialog.
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: Language

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

Gedae, Inc.
www.gedae.com
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
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.
What is Gedae?

.. that Gedae **transforms** under user control...

User can set optimization parameters that are independent of the graph to guide transformation
What is Gedae?

...to operate efficiently on a virtual machine.

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

Gedae’s Structure

- The block diagram is transformed using over 100 algorithms.
- The transformations establish the:
  - Order of execution
  - Queue sizes
  - Granularities
  - Memory layout
  - Dynamic schedule parameters
  - Data transfer types and parameters
  - Mode control

The Gedae transformations build a detailed model of the deployed application. Gedae uses that information to provide visibility.

Gedae, Inc.
www.gedae.com