

# **An Introduction to the Processing Graph Method**

by

**D. J. Kaplan**

**March 24, 1997**

The Processing Graph Method Tool (PGMT) product is being released under the GNU General Public License Version 2, June 1991 and related documentation under the GNU Free Documentation License Version 1.1, March 2000. <http://www.gnu.org/licenses/gpl.html>

## Abstract

*The Processing Graph Method (PGM) technology is a paradigm that provides the engineer/system designer with an environment to specify and maintain applications in a form which is invariant over a wide range of MIMD message passing parallel architectures.*

*We will give the reader the flavor of this approach by providing a description of an application; illustrating how that application is represented; showing typical results obtained by running the application; providing a short description of the atoms that are used to build the application; and providing a short discussion of the technical issues to which PGM must respond.*

## Introduction

The Processing Graph Method (PGM) is a data-flow graphics design and programming technology invented at NRL. PGM is used by system builders to build affordable applications. PGM provides a framework with which engineers can build applications that are independent of the parallel-processor architectures on which they run. The development of PGM for automated programming of parallel signal processors has led to a natural, highly effective user interface and efficient programming technique. This has been demonstrated on the AN/UYS-2, the VAX, and SUN series of workstations.

The Processing Graph Method uses an iconic description to cut parallel-processor software costs and schedule. Because Processing Graphs are independent of parallel computer architectures, The intention is to eliminate the need to rewrite programs when upgrading from one parallel computer hardware architecture to another. Because a Processing Graph is iconic, closely resembling a block diagram, scientists and engineers quickly learn the technique and are able to produce good programs.

This work was partially supported by the Office of Naval Research Computer Science Program.

The Processing Graph Method is a technology to facilitate the architecture independent programming of MIMD systems with a small to large number of processing elements. The PGM approach is intended for general system building, including signal processing. PGM has emphasized the signal processing domain because implementing signal processing systems depends on successfully programming multiple processing element

systems. The systems supported by this technology are meant to encompass differing heterogeneous processing elements that are connected in a fashion ranging from low to high communication bandwidths and from small to large time latencies. The initial funding and implementation of PGM has been through acoustic signal processing systems.

PGM now supports C4I's front end process, signal processing. The next step for PGM in C4I is to support the building of a high-tech moving picture version of a plot board. To apply PGM to that problem we must identify and implement the fusion-center multisensor correlation primitives that will support the building of such a plot board.

The Processing Graph Method Tool (PGMT) is a high performance computing software development environment. This environment helps translate an engineer's design of a weapon system into software for the weapon system's computers and this approach is specifically applicable to massively parallel distributed-computer processing.

For the application software developer, PGMT will provide:

- earlier and less costly development of new parallel processor software;
- faster modernization of parallel-processor software and earlier
- availability of enhanced performance features;
- lower costs for modernizing software;
- lower software maintenance costs; and
- more uniform capabilities across parallel-processor architectures with
- more compatible software for various hardware configurations.

“<http://www.ait.nrl.navy.mil/pgmt/pgm2.html>” is the PGMT homepage.

## Background

Fig. 1 depicts PGM's maturation over twenty years.

Produce an approach to easily program the AN/UYS-1 also called the Advanced Signal Processor or ASP. This resulted in A common Operating System (ACOS).

Define the language specification for a new multi processor standard navy signal processor, the AN/UYS-2 also called the Enhanced Modular Signal Processor or EMSP. This resulted in the Processing Graph Method (PGM) which was used as part of the EMSP specification.

Design and build a set of software tools to allow PGM to be affordably and efficiently used on a wide range of parallel processors so that applications, once built, for a parallel processor could be reused with little cost.

**Figure 1. Processing Graph Method Time Line**

**PGM technology**

We define a processing system to be processing elements linked by data transfer components. At present when a processing system changes, applications written for that processing system must be reconstructed at great cost. PGM is designed to change this.

PGM includes three components:

- A program description paradigm that provides the engineer or programmer the tools to specify and maintain applications in a context that is independent of processor architecture.
- A processor analysis system that allows a hardware designer and manufacturer to describe the way the processors and memory resources that compose a parallel processor are connected.
- An environment that will cause applications written in the program description paradigm to execute efficiently on hardware whose architecture is described by the processor analysis system.

Characteristics of PGM are:

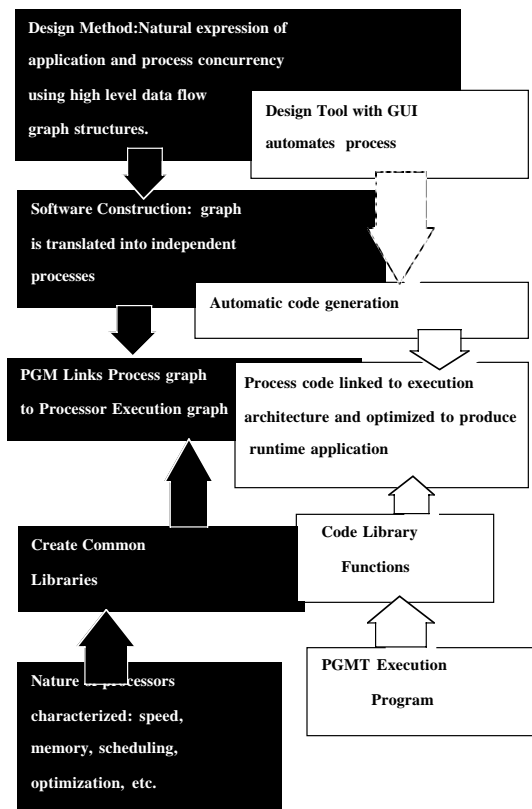
- application must be characterized only once.
- system change is only in processor portion
- New processors must be characterized only once.
- Libraries under this paradigm enable existing applications to run the new system

Navy programs using PGM are NRaD/Hughes Ground Systems on the SURTASS Project; Westinghouse/Honeywell on the AN/SQQ-89 Project; AT&T, MCCI, JRS, and GE(Lockheed Martin) on the ALFS Project; NAWC uses PGM and NUWC will alpha test PGMT; ARPA’s RASSP Contractors are using PGM.

Industry involvement with PGM includes CNA who suggested that PGM become a software standard for embedded processors; Dynetics, Inc who marketed a subset of PGM as DataFLO; Lockheed Martin through RASSP; Lockheed Martin through GE, IBM Manassas and LORAL Rolm acquisitions; MCCI which was founded to develop PGM technology; and JRS Laboratories who uses PGM technology with ARPA/AF/ARMY/RASSP.

NRL is identified by computer manufacturers as the center for this technology and PGM 2.0 is being standardized by IEEE.

Fig. 2 indicates the process flow of PGM ideas.



**Figure 2. Process flow of PGM process**

**The Processing Graph Method Tool (PGMT)**

PGMT is the environment necessary to allow PGM to be used on a processor network.

The PGMT Project Schedule time-table is:

- 1996 Completed PGMT Environment demo: GUI tool for process design, scheduling system algorithms for a single processing element machine for signal processing applications
- 1997 (end-year) Demonstrate PGMT for MIMD machines: capture hardware/operating system

architecture, more work on data structures, develop static scheduling algorithms

- 1998 PGMT with Dynamic Scheduling on MIMD: define scheduling algorithms that permit a task to be assigned to different processing elements
- 1999-2000 C2 Functions: develop command and control primitives, implement libraries and refine PGMT to date.

PGMT goal measure of effectiveness goal is to obtain maximum throughput under user-specified latency requirements.

Since this is an np-complete scheduling problem, heuristics will be used to obtain a suboptimal solution.

The PGMT project's exit criterion is to obtain results consistent with 80% of what a skilled human programming team could produce.

### A Processing Graph Example

The example we will discuss here is in the domain of signal processing. The processing graph constructed in this example is called gramGraph. The gramGraph receives sampled time-series acoustic signal data taken from a linear array of hydrophones and outputs the signal spectral energy vs time in LOFAR Gram format (x-axis:frequency, y-axis:time and z-axis-intensity:energy level) for a selected beam azimuth direction with respect to the array.

This is often referred to as narrow-band processing with beamforming and is a widely use basic process in acoustic surveillance systems. The processing graph accepts user selections for various process parameters including frequency resolution, frequency range and beam direction.

The application, gramGraph, proceeds as follows:

- Demultiplex, select and package input data samples into blocks consistent with frequency resolution and overlap selections. Fan out into a *family* of graph processes, one member for each hydrophone in the array.
- Transform to frequency domain and apply windowing and shading weights to all *family* members in parallel.
- Fan in and beamform the hydrophone spectral data for the selected beam direction and frequency range.
- Convert and equalize the beam spectral data for the LOFAR Gram display and output.

This processing graph is made of several kinds of nodes. Later in the paper these nodes will be described. For now, here is a node parts list:

- Component Count
  - 121 Queues
  - 25 Graph Variables
  - 75 Transitions
  - six Graph Input Ports of which five are for operator control data and one is for input data from sensor
  - one Graph Output Port
- Distinct Transition Types
  - three are defined and implemented as part of the PGMT specification; they are FANIN, FANOUT and PACK
  - nine are defined and implemented specifically for use in gramGraph.

Fig. 3 is gramGraph.

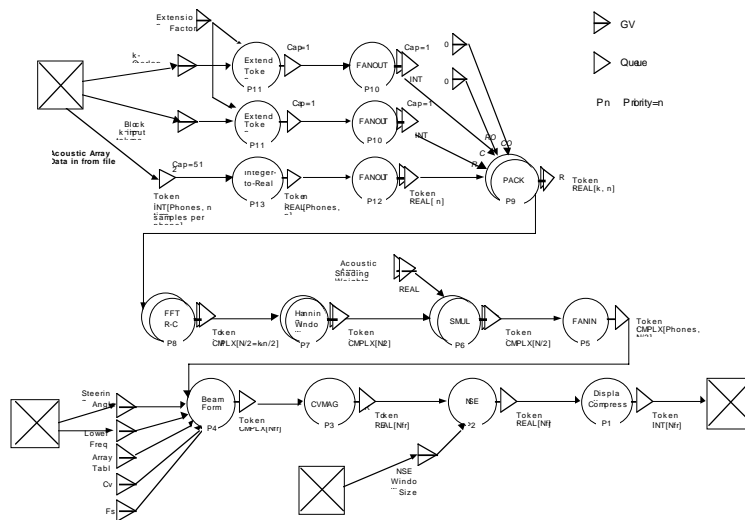


Figure 3. gramGraph

Fig. 3 is a flat structure, it has no hierarchical structure. In fact PGM is capable of expressing a hierarchical structure. To see how this is done we first construct Fig. 4

In order to use gramGraph we produced an operator interface illustrated in Fig. 7.

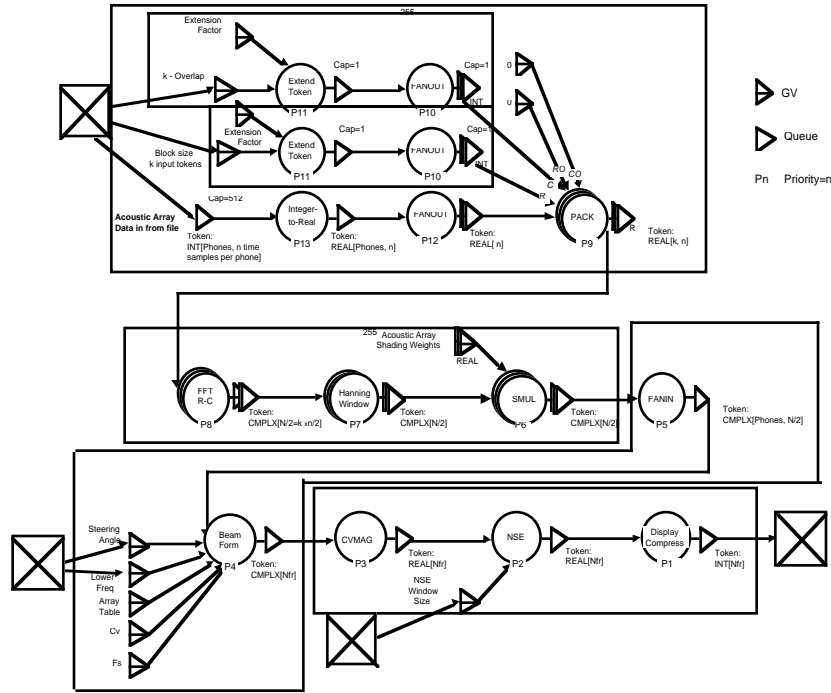


Figure 4. gramGraph (segmented)

In fig. 4, each segment is named in boldface and placed within a rectangle-like shape.

We now abstract the hierarchical structure contained in Fig. 4 and produce the tree shown in Fig.5.

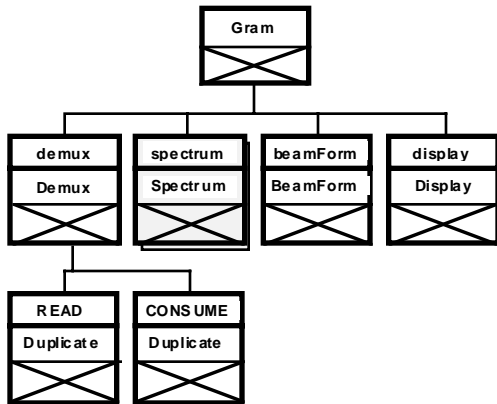


Figure 5. gramGraph tree

In Fig. 5 a segment named "family of spectrum" is highlighted. Fig. 6 depicts a single graph that supports one of the members of the spectrum family.

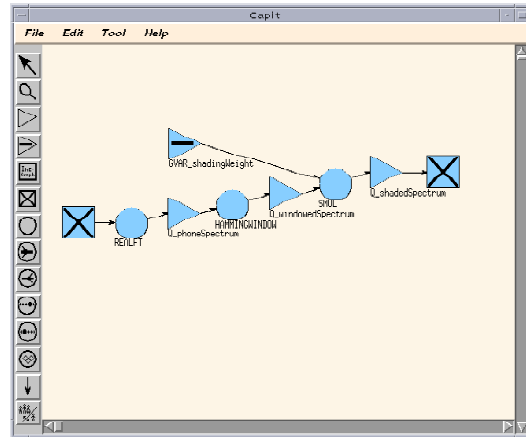


Figure 6. Member of spectrum family

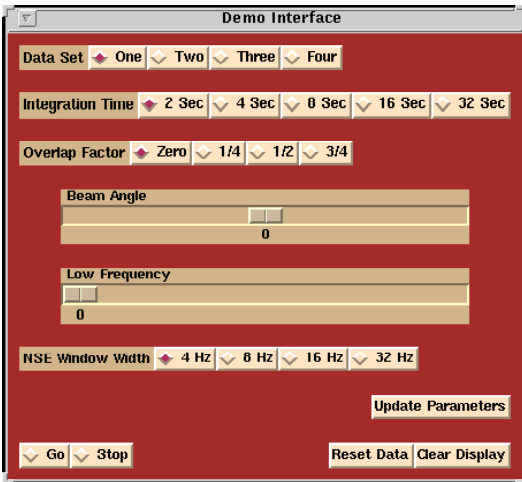


Figure 7 gramGraph Operator Interface

When this system is run, using recorded open-ocean data the following LOFAR Gram shown in Fig. 8 resulted:

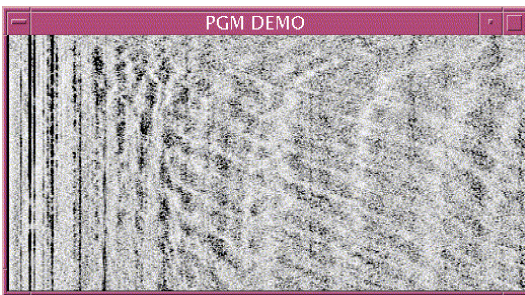


Figure 8. Demonstration LOFAR Gram

This concludes our examination of gramGraph.

### Processing graph semantics overview

A processing graph application is constructed from Command Programs, nodes (which may be places or a transitions), arcs, transitions statements, and primitives.

### Command Programs

A command program is written in a high level language. Its function is to construct, configure, control, and interact with a processing graph.

A processing graph is constructed by calling an instantiation procedure (IP). An IP calls a graph procedure to include a node (place or transition) in a graph; initialize (place or transition input port); connect a pair of ports. An IP calls another IP to construct an included graph. Command Program may call IPs. Among graph procedures are start and suspend graph; unlink a linked pair

of ports; delete a node or included graph; save a suspended graph and subsequently reload it; read tokens from a graph output place port; write tokens to a graph input place port.

### Processing graphs

Processing graphs, as gramGraph indicates, are composed of various nodes connected by directed arcs. The nodes that constitute a processing graph are show in Fig. 9.

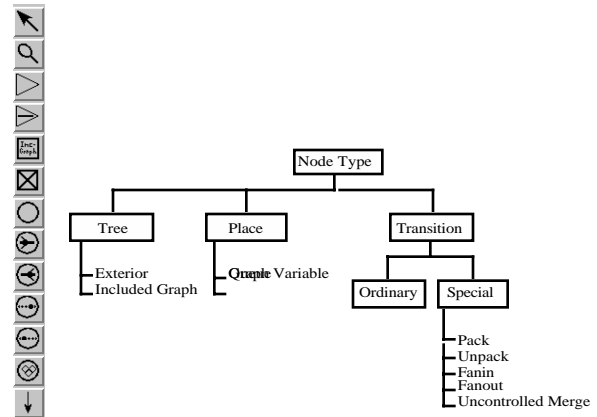
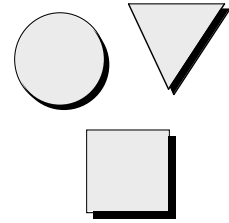


Figure 9. Processing Graph Icon Taxonomy

On the left edge of Fig. i are the icons that represent the listed nodes. We will indicate the pairing as we discuss each icon.

### Familys of graph parts

Any iconic part of a processing graph application may occur in *familys*. A *family* is special kind of structure that resembles a list limited to some basetype. The parts that may constitute *familys* are places, transitions, included graphs, and graph ports. Directed arcs are not really objects and don't make up *familys*



However, two *familys* of ports may be connected pairwise. Families are indicated by icons with dropped shadows as indicated.

### Determinacy

Please note that when we say that a processing graph has the property of *determinacy* we mean that output token streams depends solely on input token streams, independent of order of execution of transitions, and independent of all time considerations. We note, without proof, that

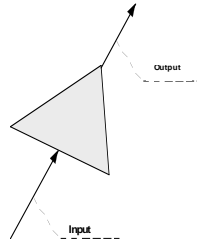
determinacy can be lost when a graph variable is read and written by two different transitions, and by any use of uncontrolled merge

**Directed arcs and ports.** A directed arc connects two ports. Ports have three attributes, two of which have binary valued attributes. These attributes are direction which has the value “output” or “input”; and category which has the value “transition” or “place”. The third attribute is mode; mode is the type of the token that goes through the port. For an arc to connect two ports, the ports must be of opposite direction, opposite category and same mode. An arc is represented by an arrow from output port to input port. Tokens pass from output port to input port.

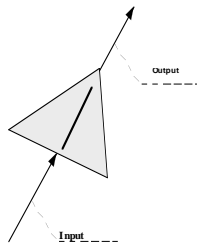


**Places.** There are two kinds of places, queues and graph variables. Places capture state.

Queues store a *family* of tokens first-in-first-out. Each queue has an attribute called capacity which is the maximum content of its associated token *family*. There is at most one transition port connected to each queue port.

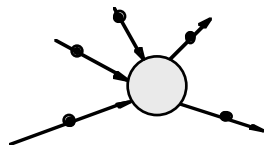


Graph variables store a single token. The value of the old token is destroyed when a new token is supplied. The value read is the most recent token. Any number of transition ports can be connected to each port.



**Transitions.** There are ordinary transitions and special transitions. There are several kinds of special transitions. Transitions capture function.

The most frequently encountered transitions are ordinary transitions. Ordinary transition ports are simple. There are no *families* of ports and one token is read and consumed at each input port followed by an associated transition statement being executed. Please note that embedded within the transition statement may be any number of primitive functions. After the transition statement is executed, one token produced at each output port.

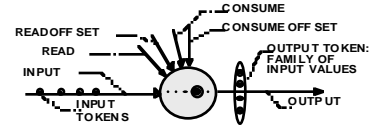


The remaining transitions are classified as special.

The first special transition we will consider is the pack transition. The pack transition has an input port named INPUT which may be of any mode.

The Pack transition has four other input ports:

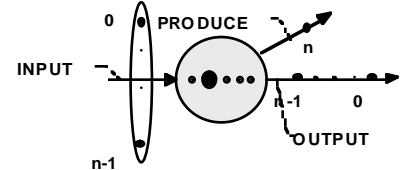
- READ of mode int,
- READ OFFSET of mode int,
- CONSUME of mode int,
- CONSUME OFFSET of mode int,



The Pack transition has one output port named OUTPUT whose mode is *family* of INPUT mode. When a Pack executes it takes a number of tokens, as specified by READ and READ OFFSET, from INPUT. Pack takes those tokens and places them in a single token on OUTPUT. It then consumes tokens from INPUT as specified by CONSUME and CONSUME OFFSET.

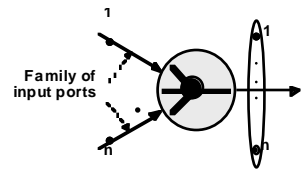
The Unpack transition has an output port named OUTPUT that may be of any mode. It has another output port named PRODUCE of mode int. Unpack has a single input port named INPUT whose mode is “*family* of OUTPUT mode”.

Unpack accepts a token which constitutes a *family*. It unpacks that tokens and sends the resultant tokens to OUTPUT. It places on PRODUCE the number of tokens that were in the input token. Please note that the content of the input tokens may be zero.

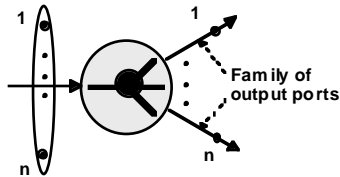


A Fanin transition has as input a single *family* of input ports; these ports may be of any mode and that mode is common to all members of the *family*. The transition has one output port whose mode is “*family* of mode of input port”.

Execution may take place when each member of the input *family* has at least one token and the capacity of the output place is at least one greater than its content. One token is read and consumed from each input port. One token is produced at the output port. That token is a *family* constructed from the read tokens and indexed in the same order as the index of the *family* of input ports.

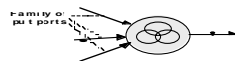


The Fanout transition has as output a single *family* of output ports which may be of any mode and that mode is common to all members of the *family*. The transition has one input port whose mode is “family of mode of output port”.



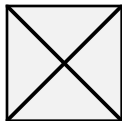
Execution may take place when the input port has available to it at least one token and the capacity of each output place is at least one greater than its content. One token is read and consumed from the input port. One token is produced to each member of the output *family* of ports. Those tokens are obtained by disassembling the input token into its individual subtokens. The indices of the subtokens are the same as the indices of *family* of output ports.

An Uncontrolled Merge has a *family* of INPUT ports which may be of any mode. It has one OUTPUT port of the same



mode as the *family* of INPUT ports. To execute, there must exist at least one INPUT token on some INPUT port, and there must be space for one OUTPUT token on the output. Execution consists of one INPUT token being read and consumed and one token produced to OUTPUT of the same value as the INPUT token.

**Graph exterior.** A Graph exterior is an icon which serves as the interface to the world exterior to the processing graph. Its semantics are that it consists of the graph ports of the processing graph, where each graph port has the attributes of Mode, Direction (with the possible values of “input” or “output”) and Category (with the possible values of “Transition”, “Place”).

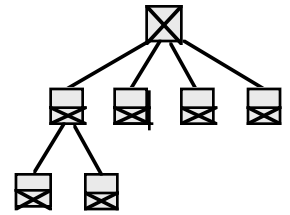


**Included graph.** An Included graph is an icon which represents the interface between an embedded graph and the processing graph which supports it. Its semantics are that it consists of the graph ports of the embedded graph. For every Included graph there is a corresponding Graph exterior. The processing graph which has that Graph exterior is called the support graph of the Included graph. For every port in an Included graph, its direction is opposite the direction of that port in the corresponding Graph exterior. For every port in an Included graph, its mode is opposite the direction of that port in its Support graph. For every port in an Included graph, its



category is opposite the direction of that port in the Support graph. For every port in an Included graph, its mode is the same as that port in the Support graph.

**Graph tree.** In the example shown earlier, the topmost graph was the root of a tree and is referred to as the main graph. In our example the Main graph had three included graphs and a *family* of included graphs. One of the Main graph’s included graphs had two included graphs. Two of included graphs had the same support graph.



## Summary

The Processing Graph Method (PGM) technology is a paradigm that provides the engineer/system designer with an environment to specify and maintain applications in a form which is invariant over a wide range of MIMD message passing parallel architectures.

We gave the reader the flavor of this approach by providing a description of an application; illustrating how that application is represented; showing typical results obtained by running the application; providing a short description of the atoms that are used to build the application; and providing a short discussion of the technical issues to which PGM responds.

## References

- [1] D.J. Kaplan and R.S. Stevens, *Processing Graph Method 2.0 Semantics*, U.S. Naval Research Laboratory, 1995
- [2] Erner, K.A. and Gorline, J.L., *The Processing Graph Method Tool C++ Reference Manual*, U.S. Naval Research Laboratory, 1996.
- [3] Erner, K.A. and Gorline, J.L., *The Processing Graph Method Tool C++ Programming Tutorial Version 2.0*, U.S. Naval Research Laboratory, 1996.
- [4] Gilbert Christopher Sih, *Multiprocessor Scheduling to account for interprocessor communication*, University of California, Berkeley, Memorandum No. UCB/ERL M91/29, 22 April 1991.
- [5] Richard M. Karp & Raymond E. Miller, Properties of a Model for Parallel Computations: Determinacy, Termination, Queuing, *SIAM J. Appl. Math.* v14, pp 1390-1410, 1966.
- [6] J. Aylor, R. Klenke, and R. Hillson and D. J Kaplan, VHDL-Based Performance Modeling for the Processing Graph Method Tools (PGMT), *Proceedings of the VHDL International Users Forum* (VIUF Spring 1996), Feb 28 - March 2, Santa Clara, CA.