Toward Embedded Development from Advanced Khoros

Joe Fogler
Albuquerque High Performance Computing Center
1601 Central Ave. NE
Albuquerque, New Mexico 87131
fogler@arc.unm.edu

Tom Robey and Mark Young
Khoral Research Inc.
6200 Uptown Blvd. N.E., Suite 200
Albuquerque, New Mexico 87110
trobey@khoral.com, young@khoral.com

Abstract

Current practice in the design of application software for high-performance embedded computing systems is characterized by long development times, lack of interoperability with other systems, and handcrafting using minimal tools. In this paper we propose a solution to the problem based on the Advanced Khoros software environment. Significant enhancements to Advanced Khoros are being implemented in support of embedded computing. These extensions include support for streaming data flow, differentiation in the behavior of interprocess communication across different application domains, and the compilation of graphical program descriptions to embedded systems.

1   Introduction

Embedded computing systems are encountered in a variety of commercial and military applications, and are utilized where limits are imposed on size, weight, and power. Operation is generally turn-key with little or no intervention by human operators. Throughput demands can be very high, particularly for military applications. In such instances, complex designs utilizing large numbers of high-performance processors may be required to meet throughput demands.

Current practice in the design of application software for high-performance embedded computing systems is characterized by long development times, lack of interoperability with other designs, and software integration by handcrafting using minimal tools. The objective of our ongoing effort is to develop software tools to improve the efficiency and reduce the cost of performing this difficult task, and to increase software lifetimes through portability. Our approach is to develop new infrastructure for the *Advanced Khoros* software environment, and to foster the development of support toolboxes by embedded systems vendors, to provide a unified environment for software development through the entire life cycle of an embedded system.

1.1 Current Advanced Khoros Development Environment

An initial step in the software development process for an embedded system is algorithm development and testing. This usually involves the collaboration of a group of scientists who prove out their algorithm concepts on scientific workstations and then convey the algorithm descriptions to a second group of individuals who implement real-time versions of these algorithms on embedded hardware. *Cantata*, a visual programming language within the Khoros software environment, provides a unique combination of capabilities for collaborative algorithm development and has gained a large following in the defense research community. Within Cantata, users can construct visual programs by connecting *glyphs*, which represent the actual processing operators to be executed, with *data connections*, which represent the data flow between these operators. Khoros provides the infrastructure to allow glyphs developed by different individuals or organizations to be easily connected. A collection of interconnected glyphs that form a complete visual program is called a *workspace*. Workspaces can be loaded and stored in Cantata using simple pull-down menu commands. This intuitive approach provides a simple and accessible way to interconnect and execute many different programs.
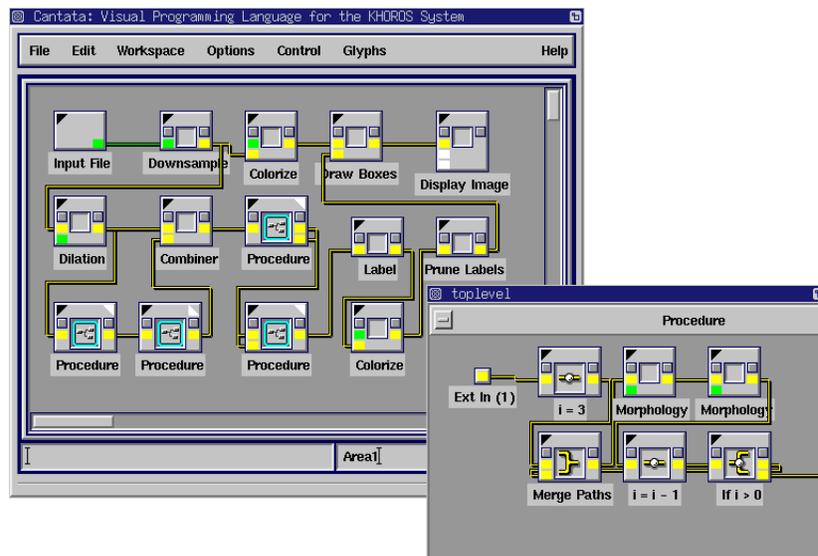

Fig. 1. Glyphs and procedures in a Cantata workspace.

Although Advanced Khoros is a powerful tool for algorithm development and testing, there is presently no built-in facility to assist in the migration of algorithm descriptions to embeddable computing resources.

1.2 Extensions to the Advanced Khoros Model for Embedded Development

The current Advanced Khoros model supports distributed processing on scientific computing platforms running the Unix operating system and possessing significant hardware resources such as large system memory and disk storage. In this model, data flows between each Khoros process (represented by glyphs in Cantata) via a transport mechanism that can represent a data file, shared memory segment, or socket connection. Interprocess comunication operations are performed as data is moved between each individual glyph. A chain of interconnected glyphs can be grouped into a *procedure* that collapses the graphical description of the chain into a single *procedure glyph*, while maintaining connectivity with other glyphs within the same workspace. This conveniently hides the detailed description of the glyph interconnections within the procedure in a lower visual hierarchy.

This model is flexible and powerful and well suited to the scientific computing environment. However, it imposes too many demands on limited resources for embedded applications. This has motivated the development of extensions to the Advanced Khoros model that target embedded development.

Streaming Data Services *Streaming data services* provides a mechanism for the continuous transmission of data messages from one data processing routine to another in Khoros. Designed to have minimal overhead for the processing of data in real-time, it represents an ideal transport mechanism to use in time-critical applications involving Cantata visual programs.

Streaming data services also serves as a model for the design of software for interprocess communication on embedded platforms using current vendor IPC middleware and will serve as a unifying abstraction software layer. This represents a near-term solution that would transition to vendor supplied real-time message passing interface (MPI/RT) middleware when available [1].

Workspace Compiler A key enhancement to Advanced Khoros focused on embedded software development is a new ability to compile workspaces. That is, an algorithm description expressed as a Cantata workspace can be compiled as one or more standalone executable programs. An extension to this facility is the capability of cross-compiling to different computing platforms utilizing existing Khoros CASE tools in conjunction with third-party compilers and linkers.

Domains A new concept is being introduced to Advanced Khoros called *domains*. Domains represent a mechanism whereby graphical programs, captured in the form of Cantata procedures, can exhibit different behaviors within different application domains. Domain examples include batch-parallel, embedded systems, and traditional Khoros visual programs.

A behavior specific to the embedded systems domain is the promotion of interprocess communication (IPC) operations in compiled workspaces from individual glyphs to Cantata procedures. This allows individual glyphs to represent

operators at the level of library function calls (primitives) that share memory space within a process. Thus, a compiled procedure represents a sequence of library primitives within a single process and IPC occurs between procedures rather than individual glyphs. This reduces the amount of IPC which increases the efficiency of code compiled for the embedded platform.

The modified IPC pattern only occurs when the code is compiled. Before compilation, the Cantata workspace on the host platform retains the standard IPC patterns of a standard Cantata graphical program with each glyph representing a separate process.
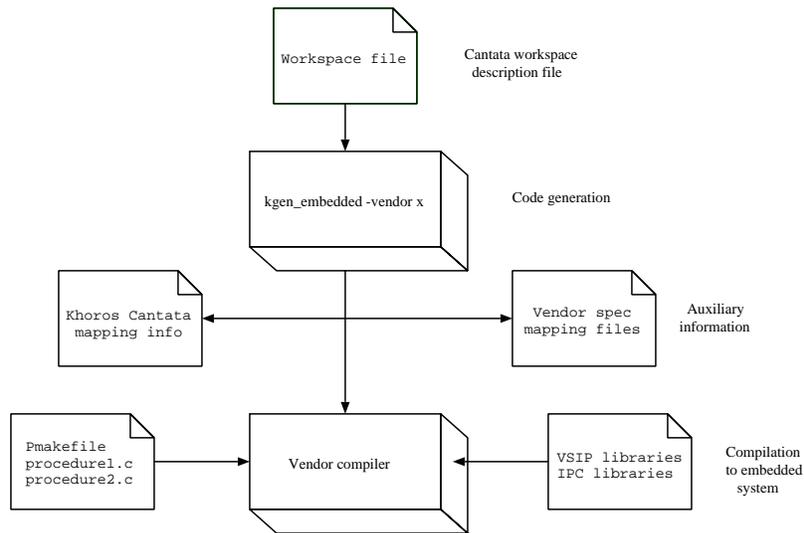

Fig. 2. Advanced Khoros model for Embedded Computing.

2 Utilization of Standardized Software Components

DARPA and other government sponsors are supporting several efforts to standardize software components for embedded applications. Khoros will incorporate such components as they become available to allow maximum software portability and minimize development costs.

2.1 Vendor Math Libraries

Embedded computing vendors generally supply scientific math libraries with their software development environments. Examples include the Scientific Algorithm Library (SAL) from Mercury Computer Systems, the Standard Math

Library from Sky Computers, Inc., and the Industry Standard Signal Processing Library (ISSPL) from CSPI [2,3,4].

Scientific library routines originated by Floating Point Systems, Inc. have served as a general guide for the syntax of vector and matrix library routines for embedded systems, but the lack of an official standard has resulted in vendor implementations with little interoperability. This situation has prompted DARPA to support the development of a standard API for scientific library routines called the vector, signal, and image processing (VSIP) libraries [5].

2.2 VSIP Libraries

A collection of vector, signal and image processing (VSIP) software libraries are being developed, representing a standard API for algorithm primitives. A reference version of the core functionality of these libraries will be available in the summer of 1998. Vendors are expected to utilize the reference version as a guide for the development of tailored versions tuned to their own hardware architectures. The VSIP standard supports the concept of public and private memory. Private memory allows vendors to exploit memory hierarchy and other computer architecture–related details in their VSIP implementations while hiding these complexities from the user. Deferred execution is also supported which allows other forms of optimization.

VSIP Parallelization Issues The current reference VSIP libraries are essentially serial implementations of selected scientific library functions. As such, applications using VSIP are restricted to a limited set of parallel modalities. Of course, task parallel operations can be performed using any VSIP operators. However, data parallel operations can only be performed by a subset of VSIP operators whose algorithmic structure allows the independent processing of distributed data. VSIP routines that require communication between distributed data must be eventually re-written to support data parallel operation.

Interoperability with Vendor IPC Middleware These are potential performance optimization problems when VSIP operators are interfaced to non-VSIP software such as operating system calls and vendor IPC middleware. Such software cannot be expected to support the VSIP private memory model unless it is specifically designed to do so. Public memory, however, is managed by the user and represents a safe mechanism by which data can be imported or exported from VSIP. Consequently, data must be continually copied between buffers in private and public memory at the interface between VSIP and other software which results in significant overhead. Vendor IPC middleware may become available at some point that is fully compatible with the VSIP private memory model thereby eliminating the need for extensive buffer copying.

Interoperability with Khoros Public memory is used in Khoros glyph implementations of VSIP reference operators at the interface between VSIP library

routines and Khoros streaming data services calls. In the reference implementation of VSIP, the user is required to bind a public memory buffer to a *block object* if it is re-located or interchanged with another buffer. This results in overhead as block objects are created and destroyed. A *re-bind* function has been proposed in VSIP to allow buffers to multiplex with a single block object by simply updating a pointer entry in the object. This proposed function is complementary to and cannot replace the proposed VSIP functions for data import and export.

As a first implementation of cross-compiled Cantata procedures representing chains of VSIP primitive calls, public memory will be used to interface vendor IPC objects to VSIP operators. These interfaces will occur at the inputs and outputs of procedures. VSIP primitive calls that only communicate with other VSIP primitives within each procedure will utilize private memory in order to exploit vendor optimizations. This initial implementation may be refined later to utilize private memory throughout if vendor IPC objects can support it. However, a mechanism will be needed to override deferred execution at branch points in the overlying Cantata visual program.

2.3   Vendor Interprocess Communication Middleware

Parallel distributed computing on embedded systems requires the use of real-time operating system kernels and libraries, and IPC software. The operating system and IPC software might not be available from the same vendor but must have a high degree of interoperability.

Vendor IPC middleware can be expected to include IPC objects such as shared memory buffers, semaphores, and in many cases, sockets. Although there are differences among vendor implementations, a useful degree of commonality can be found. The initial implementation of compiled workspaces for embedded platforms will utilize a software layer that maps the process and data flow of the Cantata visual program onto vendor IPC middleware.

2.4   Transition to MPI/RT

A standard for interprocess communication in real-time called MPI/RT is currently being developed. The goal of MPI/RT is to provide the middleware for programmers to create real-time applications with performance portability, leveraging experience with existing Message Passing Interface (MPI) software. MPI/RT will provide a consistent set of extensions (and some restrictions) to MPI while adding greater predictability and schedulability to message-passing programming. Core features are quality of service, minimal overhead, and early binding.

As vendors adopt MPI/RT as the IPC strategy for their embedded systems, the Advanced Khoros embedded support environment will be tailored to allow migration from vendor-specific IPCs to MPI/RT-based code for interprocess communication.

2.5 Mapping of Applications to Hardware Topology

Inherent in the conversion of visual programs in Cantata to efficient code on multi-processor embedded systems is the need to describe the mapping of processes and connections in the visual program to hardware topology [6].

Current Application Configuration Mapping Tools Mapping binaries to processors and messages onto the communication backbone is currently a manual process. Tools such as Mercury's Talaris Configuration Editor (TCE), Peakware, and Sky's Configurator Tool provide an interactive mechanism for generating the code required for the mapping of multiple software processes onto hardware topology. This manual mapping process is time consuming, labor intensive, and limits portability since it has to be performed for every hardware platform. Another issue is that the mapping process tends to be uni-directional.

Strategies for the bi-directional flow of mapping descriptions between Khoros and embedded environments will be explored to permit iterative software development for embedded systems from within Khoros.

Automatic Mapping Tools DARPA has funded an effort to develop an automatic mapper extension to Mercury's TCE. Automatic mapping is a non-linear optimization problem that allocates resources such as memory consumption and communication latency. This effort will look at building a framework that will allow flexible algorithm selection for the mapping process and provide a few mapping algorithms. The automatic mapper will use a graph of computational operators and connecting communication with appropriate software parameters, and map to a graph of physical hardware consisting of computational processors and communication fabric with hardware parameters such as processor speed and memory, communication latency, and bandwidth. Three proposed algorithms are a first-fit, simulated annealing, and a Global Criticality/Local Phase (a type of greedy) algorithm [7].

3 Conclusions

The development of application software for high-performance embedded computing systems is characterized by long development times and lack of interoperability with other systems. Our objective is to develop software tools to improve the efficiency and reduce the cost of performing this task.

An approach has been presented where new infrastructure is being developed for the Advanced Khoros software environment to provide a unified environment for software development through the entire life cycle of an embedded system. Advanced Khoros is being extended to support streaming data flow, differentiation in the behavior of IPC across different application domains, and the compilation of graphical descriptions of algorithm programs to embedded systems. Standardized software components including the VSIP libraries, vendor IPC middleware, and ultimately, MPI/RT are employed. Software tools are

also being investigated for the mapping of algorithm descritions onto hardware topology.

References

1. MPI/RT Forum Committee, *DRAFT Document for the Real-time Message Passing Interface (MPI/RT) Standard*, Real-time Message Passing Interface Forum, http://www.mpirt.org, 1998.
2. Mercury Computer Systems, *Developers Guide*, Mercury Computer Systems, Chelmsford, MA, 1997
3. Sky Computers, *SKYscl Users Guide*, Sky Computers, Inc., Chelmsford, MA, 1997
4. CSP Inc., *Industry Standard Signal Processing Library (ISSPL)*, CSP Inc., Billerica, MA, 1997
5. Randall R. Judd, *Vector and Elementwise Operations (DRAFT)*, Vector Signal Image Processing (VSIP) Forum, http://www.vsip.org, 1997.
6. B. Isenstein, M. Krueger, and A. Pool, *Middleware for Realtime Multicomputer Tool Development*, Mercury Computer Systems, Inc., Chelmsford, MA 1995
7. A. Kalavander, E. A. Lee, *A Global Criticality / Local Phase Driven Algorithm for the Contrained Hardware/Software Partitioning Problem*, Proc. of CODES/CASHE, Third Intl. Workshop on Hardware/Software Codesign, Grenoble, France, Sept 22-24, 1994, pp. 42-48.