Final Report: High Performance (Parallel) Object-Oriented Software Systems (HiPPO)

Prof. P.A. Lee, Dr. C. Phillips, Prof. P. Watson

School of Computing Science
University of Newcastle upon Tyne, NE1 7RU

1. Background/Context

Over many years research has addressed the problems of designing and implementing software systems. Supporting tools have been developed, but the increasingly complex demands of applications have kept the need for new tools and paradigms ahead of their supply. The object-oriented paradigm is one weapon in the software engineer's arsenal, and is seen as a highly beneficial engineering methodology, with growing tool support, whose importance is unlikely to diminish.

One area of software development that has not received the tool support it needs is the exploitation of parallelism. Generally, the programmer has been left to manage control-flow complexity, low-level parallelism primitives and parallel architecture dependencies. Moreover, one of the last areas to be affected by object-orientation has been high-performance computing. The numerical area traditionally has shied away from object-orientation - preferring to optimise efficiency at the expense of overall software engineering. However, as discussed in the original proposal, interest in o-o in these areas is also growing.

High performance isn't just a preserve of "traditional" numerical applications. Many application areas are increasing their demands for processing power. The single-cpu solution to satisfying the need for processing power is not going to continue indefinitely; exploiting parallelism is the obvious requirement, particularly as commodity hardware can provide cheap parallel platforms.

Following on from successful PhD research, the HiPPO proposal was to investigate *visual programming notations and syntax* as a way of providing support for constructing parallel, object-oriented software systems. A visual notation was seen as a way of escaping from the limitations of traditional textual notations which, when combined with an appropriate computational model, would enable the natural expression of software solutions from which parallelism could be extracted automatically.

2. Key Advances and Supporting Methodology

The principal research advances generated during the project, which will be explored in more detail in the following sub-sections, are:

1. The design of a novel visual notation and computational model for expressing parallel, object-oriented software systems, to investigate the potential of constructing implicitly parallel o-o software visually.
2. The design and implementation of a supporting IDE (Integrated Development Environment), to enable "graphs" representing programs to be constructing, and hence the use of the visual notation to be explored
3. The development of a compiler for translating HiPPO program graphs (using the into C++, utilising the NIP run-time parallel environment also developed at Newcastle, to

provide a practical environment for evaluating the above advances, as well as researching run-time issues (e.g. performance optimisation).

2.1 HiPPO Visual Notation and Computational Model

The first part of the project investigated the state-of-the-art in visual programming notations, surveying the existing work [Lee03] and considering the features needed for HiPPO together with the semantics of the underlying computational model. A number of alternatives were considered and evaluated before the final HiPPO notation set was arrived at, and it is some of the main features of this final version that will be described here.

The HiPPO language is graph-based, with boxes representing parts of a computation (e.g. method invocations) connected by arcs that control the computation. HiPPO provides a shared object environment, where even basic numbers are first class objects. A number of different box types are provided by the IDE, and can be seen on the left-hand side of the IDE screen shot in Figure 2 below. An example HiPPO program graph for Matrix multiply is provided in Figure 1.

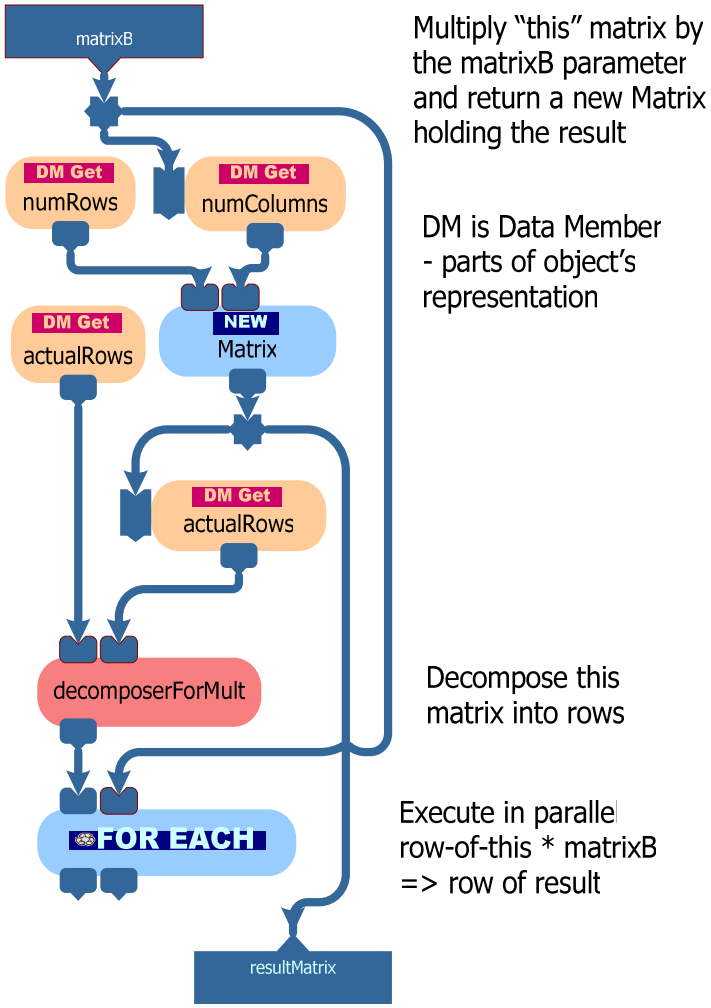

Figure 1. Matrix Multiply HiPPO graph

The key novel features of the prototype system that the project has developed are bulleted below. It is impossible to describe all of these in detail here; further explanation is available in [Lee04]:

- *References to objects* (termed object handles), rather than copies of data values, flow through arcs and trigger computations. Thus, it is the flow of object handles that primarily determines the parallelism that is available at run-time, rather than requiring the programmer to control parallelism explicitly.
- In addition, HiPPO provides control-flow arcs for specifying additional sequencing constraints on computations in those limited occasions where this is necessary (e.g. to obtain the correct sequencing of two printing operations).
- Many of the low-level synchronization requirements of the object-sharing model are delegated to the run-time system and do not require explicit manipulation by the programmer.
- The arc containing the handle for an object to which a method is to be applied flows to a special attach point at the side of a method-call box. Attach points may have non-blocking or blocking semantics (the latter inhibits transmission of the object handle until the associated method call has completed, the former permits potentially parallel use of the object).
- HiPPO supports a general, user-definable means for generating independent object handles (called decomposers), along with a special computation node (**FOR EACH**) that can processes the handles in parallel computations.

2.1. HiPPO IDE

The HiPPO Integrated Development Environment was designed and implemented to support (a) the specification of classes and their interfaces; and (b) the implementation of the HiPPO graphs representing the methods. Microsoft Visio was used as a prototyping tool to permit the rapid exploration of proposed features. Subsequently, this experience influenced changes to the HiPPO plan, as discussed below. Key novelties in the IDE include:

- Visual support for a number of features (including those found in traditional textual OO languages):
  - the definition of types (classes) and their interfaces, their member variables, and access to these features from program graphs
  - the conditional execution of graphs; two sub-graphs are used for the ‘true’ and ‘false’ execution path
  - the **FOR EACH** node; a sub-graph specifies the parallel computation
  - sequential iterative computations (the loop node) for use when computations have recurrence relationships. A loop’s implementation is defined using 4 sub-graphs for the initialisation, testing the condition, the body of the loop, and the post-iteration computation
- The IDE permits the user to provide performance optimisation “hints” about methods and their arguments – e.g. a read only property, or a “clone this object” indication can be used at run-time to optimise locking, object replication and distribution.

Figure 2 is a screenshot of the HiPPO IDE, showing the template of icons for constructing HiPPO graphs on the left, and the Class Designer window in front. This window is displaying some of the interface characteristics of the Matrix class, including the methods (e.g. Matrix (constructor), initElements(), multiply()), methods arguments and result sets. In particular, the details for the matrixB argument are expanded further in the foremost Item properties window.

HiPPO graphs such as the multiply example in Figure 1 are drawn by dragging-and-dropping icons from the template onto a window, with the IDE permitting the selection of method calls as necessary. Thus the IDE provides a rich environment for the specification of HiPPO programs.

An XML schema was designed in the project to enable XML representations of HiPPO programs to be generated by the IDE, so enabling mappings from HiPPO to different execution frameworks (currently C++ and Java are being investigated).

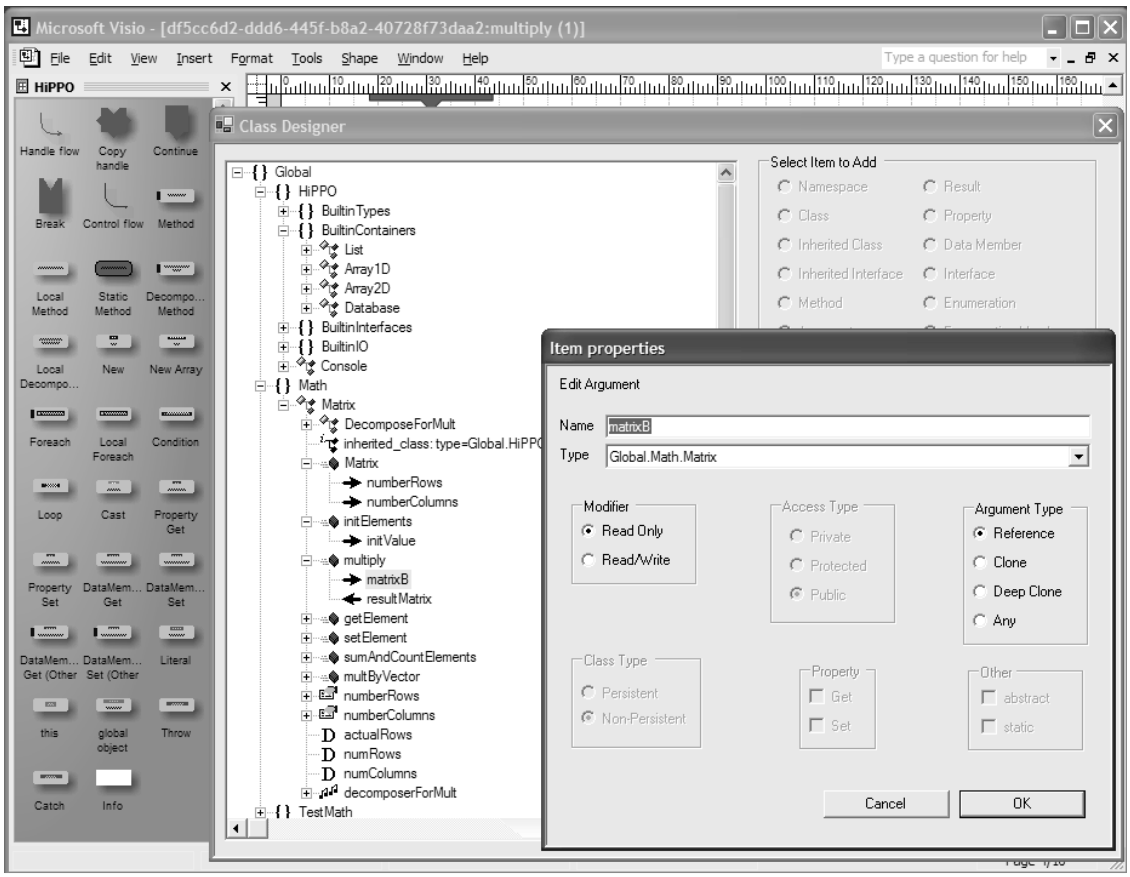

Figure 2. HiPPO IDE Screenshot

2.3 HiPPO Compiler and NIP Run-Time System

A compiler was written in Java to accept the XML files generated by the HiPPO IDE and to generate C++ code to link to NIP, a run-time system designed at Newcastle [Watson99] that provides an execution environment for implicitly parallel, object-oriented programming languages like HiPPO. NIP offers a shared-object memory space, and run-time constructs for the identification of potentially parallel computations that language compilers can use.

NIP dynamically manages the degree of parallelism exposed by applications to efficiently utilise the computational resources offered by the underlying hardware platform which may be shared- or distributed-memory based. NIP also takes responsibility for the concurrency and caching related issues that may arise through the use of the shared-object space. By targeting the NIP execution model, the HiPPO compiler is freed from having to produce code that explicitly manages parallelism and deals with architecture related issues.

3. Project Plan Review

The actual project plan deviated from the original plan in a number of areas, due to major problems encountered in the project with recruiting and retaining RAs. We took active steps to

manage such difficulties, and changed objectives and priorities to maximise the returns in the key areas of research.

The project planned for 2 RAs for a total of 72 person months. By the end of the project, 4 RAs had been employed but only for 68% of the total time (49 person months), and only one for a period of greater than a year. The retention issue was nothing to do with the HiPPO project, its work or management. A range of personal circumstances led to the key RAs leaving (Dr Parastatidis unexpectedly had to return to Greece to do his National Service; Dr Arnaud left for a higher-salary job as his wife became pregnant with twins; Dr Mukhergee had to return to India for personal family reasons). In addition, Dr Jim Webber, who was expected to play a significant role in the project (as indicated on the original proposal), left the University to join a local startup company (Arjuna) 6 months after the project started. The lack of available RA effort together with the changing and re-educating of new personnel had a major effect on what the project could produce, and resulted in changes to the original objectives. Moreover, it was difficult to attract RAs for the remaining time left on the project, even given the 9 month extension that was granted by the EPSRC.

By the end of 2002, the major item not being addressed was the HiPPO compiler. A major source expertise with the NIP system was Dr Webber, and we tried to lure him back to the project. Instead, the opportunity arose for the work to be carried out by him through a contract with the Arjuna company. We obtained approval from the EPSRC (Matthew Griffiths) to vire funds from the underspent salary heading into consumables to fund this contract, and Arjuna were contracted to develop the compiler and deliver it by June 2003. This would have given the project time to shakedown and use HiPPO to generate additional research results and papers.

Unfortunately, development of the compiler by Arjuna took much longer than planned, and usable functionality was only available towards the end of the HiPPO project. This was a major delay for our plans to use HiPPO for real applications, such as in exploring the numerical computation area. However, we have continued to employ one RA for 2 additional months to continue HiPPO development, using personal research funds. The compiler is now relatively stable, and supports the majority of HiPPO features, and some HiPPO programs can be run on shared-memory multiprocessors or workstation clusters or both. Disappointingly, though, parallel executions involving the HiPPO foreach construct are still unreliable due to implementation bugs, and the source of these problems is the subject of on-going work.

The project originally planned to use the OO database technology developed by the Polar project. This technology was not straightforward to exploit in the time available, and instead, we obtained and tried to use a commercial OO database package (Poet). However, the development problems highlighted above took priority and this objective of the project was dropped. The project also planned to re-implement the HiPPO IDE using the HiPPO notation. However, it was decided that the optimal strategy for the project was to continue to use Visio-based IDE rather than divert efforts into regenerating the visual facilities that it provided.

The final objective which was changed was in the area of performance optimisation. The HiPPO compiler adopts the strategy of generating as much potential parallelism as possible, the idea being that subsequent optimisation steps could then look at optimising the parallelism actually employed, in harness with the NIP run-time. This objective was not completed, and hence the code generated is less than optimal in its performance objectives.

Despite these severe staffing and continuity difficulties, careful management and adjustment of the project's objectives and focus has achieved the research advances highlighted in Section 2, and provides the basis for the further work described below.

4. Research Impact and Benefit to Society

Because of the development problems discussed above and not having a working system until late in the timescale, the project has been unable to generate practical results from using HiPPO and thus publish papers on those results. We have tried recently submitting papers to international conferences and workshops, but they have been rejected – the notation was generally thought to be interesting by the referees, but the lack of actual performance results was the major handicap. The project therefore has not had the research impact that it should have. However, we are confident that publications will follow the continuing work, when results can be generated.

5. Further Research or Dissemination Activities

The software developed (approximately 40K lines of source code) is such that much interesting research remains, and our plans are to continue this work. As mentioned above, we have already funded additional HiPPO development with personal funds. Student projects are exploring the mapping of HiPPO onto Java code, showing the value of developing the XML schema for representation of HiPPO programs. A new parallel run-time system is also becoming available.

The intention is to continue use of and experimentation with HiPPO and to generate some practical results, and publish papers. The use of HiPPO in the numerical computation area is already being explored, and a new run-time system is just becoming available at Newcastle which will provide an additional avenue for research into run-time object-oriented support, and performance optimisations. The key feature for the latter is getting the right grain size in balancing computation with parallelism overheads, and the HiPPO notation and its XML mapping holds much promise as a basis for such optimisations.

Initial experiences with using the HiPPO environment for development have been very encouraging – it is certainly a different and fascinating way of developing parallel programs, and the full ramifications of the flow-of-object-references model and the visual notations require further exploration and evaluation. The software is now in a position to enable such research to take place.

Our work on surveying visual programming techniques is very relevant to the development of workflow languages that are becoming an important part of the eScience/Grid research projects, and this is an area for further research. The work was discussed and referenced in the report of the main UK eScience workflow conference.

6. References

[Lee03] *Taxonomy for Visual Parallel Programming*, P.A. Lee and J. Webber, CS-TR-793, School of Computing Science, University of Newcastle, 2003.

[Lee04] *A Visual Language for Parallel, Object-Oriented Programming*, P.A. Lee, M.D. Hamilton and S. Parastatidis, CS-TR-826, School of Computing Science, University of Newcastle, 2004.

[Watson99] *The NIP Parallel Object-Oriented Computational Model*, P. Watson and S. Parastatidis, Lecture Notes in Computer Science 1602, Springer-Verlag, May 1999.