Exploiting Vector and Heterogeneous Systems

Kwok Tat Peter Au John Darlington Moustafa Ghanem Yi-ke Guo Hing Wing To
E-mail: {aktp, jd, mmg, yg, hwt}@doc.ic.ac.uk
Department of Computing, Imperial College
180 Queens Gate, London SW7 2BZ

Abstract

Programming parallel systems is difficult especially when such systems incorporate heterogeneous components. This paper describes some approaches we are developing to cope effectively with various forms of heterogeneous parallel systems.

The first part of the paper describes Fortran-90V, an extension of Fortran 90, which can support nested parallelism for expressing irregular problems more efficiently and naturally. This is especially suited to high-performance vector architectures, such as the Fujitsu VPP300. By compiling Fortran-90V to Fortran 90, the language is available for all parallel machines, provided the machines have Fortran 90 compilers.

The second part of the paper describes the latest development in SPP(X) which was introduced in PCW'95. SPP(X) is a language which can co-ordinating the activities of a heterogeneous system. Further by associating performance models with the operators of the language, it is possible to aid the resource allocation in a program written in SPP(X).

1 Introduction

This paper describes on-going work at Imperial College on developing programming environments for exploiting the next generation of parallel machines. It is likely that these will be clusters of high-performance servers, such as multi-vector architectures and parallel architectures. The first part of the paper describes extensions to Fortran 90 which enable a more general class of applications to be executed on vector architectures. The second part of the paper presents an update on the SPP(X) environment for co-ordinating heterogeneous computation. We conclude by describing how the two approaches may be used in conjunction.

2 Nested Data-Parallelism

2.1 Introduction

Data parallelism is a well-known programming paradigm for parallel computers. It offers fine-grained parallelism where the same operator is applied to all elements of a set of data simultaneously. Languages such as HPF [11], FORTRAN 90 [1] and C* [12] fall into this paradigm and have efficient implementations on SIMD and vector machines. These languages are well suited to solving problems with regular data structures such as dense matrices or regular meshes. However, this parallelism is restricted to rectangular arrays. Although this limitation simplifies the generation of efficient target code it limits the application areas for data parallelism. It is very hard for these languages to efficiently and naturally express algorithms on irregular data structures, such as sparse matrices, graphs and trees, or to express process parallel problems. In particular it is difficult to code these types of application in these languages such that they fully exploit the power of vector machines.

These problems can be solved by using *nested data parallelism* [4]. Using nested data parallelism a parallel operation can be applied to multiple sets of data simultaneously. For example, in a conventional vector machine, a summation operation can only be applied to a single vector in each time step. With nested parallelism, this operation can be applied to multiple vectors in the same time step. This is achieved by using *hierarchical* (nested) data structures, which are described in the next Section.

2.2 Hierarchical Data Structures

A hierarchical data structures is a vector which consists of other vectors of the same type. For example, [[1,2],[3,4,5]], is a nested vector which contains two different vectors of length 2 and 3. The key to implementing nested parallelism is to transform a nested vector into a flat vector thereby flattening the parallelism to a single level.

A nested or segmented vector can be described by using a flattened vector (a vector whose elements are not vectors) and a segment descriptor. The segment descriptor is used to specify the number of elements in each nesting level and the flattened vector stores the actual values. For example, the nested vector given above can be represented by the flattened vector [1, 2, 3, 4] and a single segment descriptor [2, 3]. A more complex example uses the nested vector:

```
[ [ [1], [2, 3, 4] ], [ [5, 6] ] ]
```

which can be represented by using two segment descriptors (segd) and a flattened vector (values):

```
values: [ 1, 2, 3, 4, 5, 6 ]
segd (inner): [ 1, 3, 2 ]
segd (outer): [ 2, 1 ]
```

3 Developing a Practical Nested Data Parallel Language: Fortran 90V

In co-operation with GMD-FIRST and Technische Universität Berlin, we are developing Fortran 90V [2], an extension of the Fortran 90 which supports nested parallelism. The advantage of Fortran 90V over other proposed nested data parallel languages is that it can reuse all the existing Fortran 90 source code and is easy for the user to adopt as Fortran 90 is widely accepted by the community of scientific programmers. An overview of the Fortran 90V system is shown in Figure 1.

The Fortran 90V system transforms the nested data-parallel language into Fortran 90 and FVL

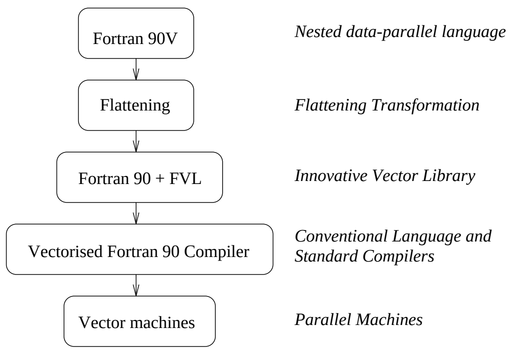
Fortran 90V
Nested data-parallel language
Flattening
Flattening Transformation
Fortran 90 + FVL
Innovative Vector Library
Vectorised Fortran 90 Compiler
Conventional Language and Standard Compilers
Vector machines
Parallel Machines

Figure 1: An overview of Fortran 90V.

by using the flattening transformation technique [9, 8]. FVL is a vector library which provides different vector operations and is directly implemented by using Fortran 90 and its intrinsic operations. This significantly improves the portability of the language as every vector machine which has a Fortran 90 compiler will automatically support Fortran 90V. A Fortran 90V compiler is currently under development. The target architectures for this compiler will include the Fujitsu AP1000 with NCA units, the Fujitsu VX, and the Cray YMP-EL.

3.1 Example - Sparse Matrix Vector Multiplication

A non-zero sparse matrix can be conceptually represented by using a nested vector containing the column number of the elements and the non-zero values. For example, a matrix such as:

$$\begin{pmatrix} 0 & 9 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \\ 6 & 0 & 4 & 0 \end{pmatrix}$$

can be represented by using a nested vector containing the non-zero values with the corresponding column indices in the matrix:

```
(< (< (9,2) >),
   (< (1,1) >),
   (< (8,4) >),
   (< (6,1),(4,3) >) >)
```

In Fortran 90V we can declare the sparse matrix as follows:

```
TYPE Element
  REAL value
```

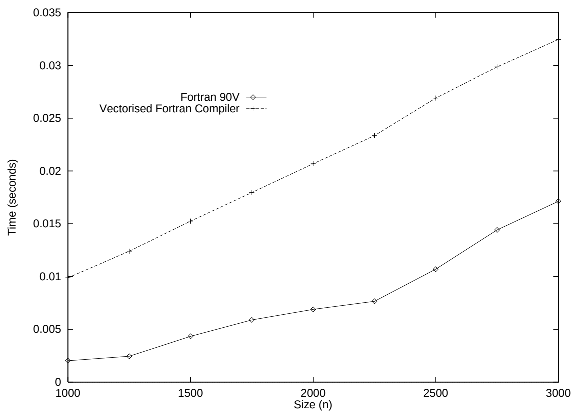

Figure 2: Performance of sparse matrix vector multiplication.

```
  INTEGER column
END Element

TYPE(Element), DIMENSION(:,:),  &
                  IRREGULAR:: matrix
```

The body of the sparse matrix vector multiplication is defined as:

```
(<sum(<elem%value*vector(elem%column) &
        ,elem=row>), row=matrix>)
```

The function first computes the product of the elements in each row with the corresponding elements in the vector and then applies the summation. Note that the constructor will compute all of the inner computations simultaneously.

As an initially investigation into the performance of this approach, the above example has been hand-compiled for the Fujitsu μVP [7] on the AP1000 at IFPC. The experiment uses a matrix size of n $\times$ n with 1% dense ratio. The results of the experiments were compared the timings for the same program compiled using a commercial vectorising compiler [10]. The vectorising compiler can only vectorise the computation in each row, but not for the entire matrix, whereas Fortran 90V can vectorise the computation across the entire matrix. A comparison of the performances is shown in Figure 2.

4 Co-ordinating Heterogeneous Parallel Computation

The idea of using SPP(X) [5] to co-ordinate the activities in a heterogeneous system was first introduced at PCW'95. Since then there has been further development on building more accurate performance models and in refining the approach [3, 6]. A prototype SPP(X) compiler using Fortran as the sequential language has also been developed although it currently lacks the placement annotations needed for complete heterogeneous co-ordination.

4.1 SPP(X) for Heterogeneous Systems

In the SPP(X) model a structured co-ordination language, SCL, is used to co-ordinate fragments of sequential code which are written in a base language (X) such as Fortran or C. Thus an application is constructed in two layers: a higher co-ordination level and a lower base level language. SCL provides high-level co-ordination forms that abstract all important aspects of parallel behaviour. These co-ordination forms are presented as higher-order pre-defined functions with known parallel behaviours. Owing to the rich structure and known behaviour of these co-ordination forms it is possible to build accurate performance models for them.

To fully exploit the power of a heterogeneous parallel computing system sophisticated resource allocation strategies are often required. This is a complex task as it is difficult to predict the performance of any particular resource allocation strategy on a system composed out of components which have widely differing functionalities and performances. Any system for programming heterogeneous systems must then be able to express the different possible resource usages and aid in deciding between different resource allocation strategies. The SPP(X) approach can satisfy both these needs by providing a high-level structured language for expressing resource decisions and performance models for guiding the choice of resource strategy.

4.2 Developing Models for Application Level Skeletons

A performance model for a program written in SPP(X) consists of two components, the model for the SCL code and the models for the sequential code fragments. A model for the SCL code can be developed from the existing models of the co-ordination forms and from the syntax of the SCL code. The remaining problem is to generate performance models for the sequential code used to instantiate the SCL skeletons. There are several techniques for finding these models, some of which are automatic. However, this time-consuming task must be performed for each program.

The level of abstraction for programming can be raised by developing application-level skeletons. For example, a suite of linear algebra skeletons such as dot product could be developed. These can be defined in SPP(X) by combining SCL skeletons to express the parallel co-ordination and by providing the sequential code to be executed on the processors. Performance models for the application-level skeletons can then be derived from the performance models of the SCL skeletons and by developing models for the sequential components. By expressing a program in terms of application-level skeletons, not only is a program clearer, but the existing performance models can be reused.

The conjugate gradient example was re-written with a set of application-level skeletons with associated performance models. The predicted performance of three different versions of the program, using only scalar processor, only vector processors and using a mixture of vector and scalar processors is shown in Figure 3. This indicates that the mixed resource version will perform best. The experimental result from hand-compiled versions of these programs is shown in Figure 4. This shows that the predictions are sufficiently accurate to enable the correct resource allocation strategy to be chosen.

5 Conclusion

In this paper we have briefly described on-going work at Imperial College on developing systems for exploiting vector and heterogeneous systems. The development of Fortran 90V will enable a broader class of applications to exploit vector architectures. Furthermore Fortran 90V is highly portable using existing standard software.

SPP(X) allows the structured co-ordination of heterogeneous systems with performance models to guide the choice of resource strategy. An in-

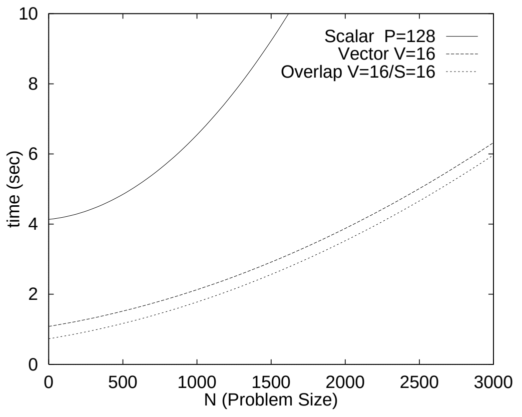

Figure 3: Predicted performance of the different programs.

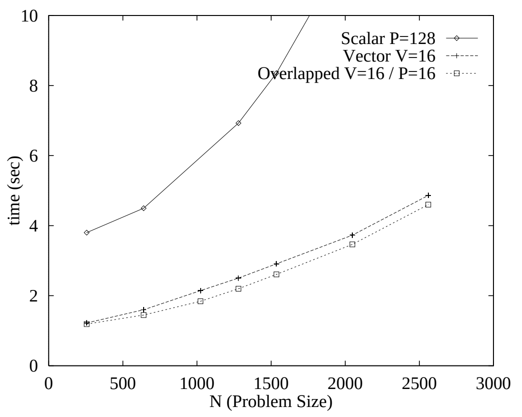

Figure 4: Measured performance of the different programs.

teresting future direction would be to use Fortran 90V as the base language for programming the vector components of an SPP(X) program. The structured approach of Fortran 90V would enable more accurate performance predictions to be made for the vector code.

Acknowledgements

The authors gratefully acknowledge support from Fujitsu Laboratories, the EPSRC funded project GR/K69988, the Croucher Foundation (Hong Kong) and the British Council. We would like to thank Fujitsu for providing the facilities at IFPC, which made this work possible. Part of this work was in co-operation with GMD-FIRST and Technische Universität Berlin, Germany.

References

[1] Jeanne C. Adams, Walter S. Brainerd, Jeanne T. Martin, Brian T. Smith, and Jerrold L. Wagener. *Fortran 90 Handbook Complete ANSI/ISO Reference.* McGraw-Hill, 1992.

[2] K.T.P. Au, M.M.T. Chakravarty, J. Darlington, Y. Guo, S. Jähnichen, M. Köhler, G. Keller, W. Pfannenstiel, and M. Simons. Enlarging the scope of vector-based computations: Extending fortran 90 with nested data parallelism. In *International Conference on Advances in Parallel and Distributed Computing*, Shanghai, PR China, Mar 1997. IEEE Computer Society Press. (To appear).

[3] Peter Au, John Darlington, Moustafa Ghanem, Yi ke Guo, Hing Wing To, and Jin Yang. Co-ordinating heterogeneous parallel computation. In Luc Bougé, Piere Fraigniaud, Anne Mignotte, and Yves Robert, editors, *Euro-Par'96 Parallel Processing*, volume I, pages 601–614. Springer-Verlag, August 1996.

[4] Guy E. Blelloch. *Vector Models for Data-Parallel Computing.* MIT, 1990.

[5] J. Darlington, Y. Guo, H. W. To, and J. Yang. Functional skeletons for parallel coordination. In Seif Haridi, Khayri Ali, and Peter Magnussin, editors, *Euro-Par'95 Parallel Processing*, pages 55–69. Springer-Verlag, August 1995.

[6] John Darlington, Moustafa Ghanem, Yike Guo, and Hing Wing To. Guided resource organisation in heterogeneous parallel computing. *Submitted to the Journal of High Performance Computing*, 1996.

[7] Fujitsu Limited. *VPU (MB92831) Functional Specifications*, 1.2 edition, Mar 1992.

[8] Gabriele Keller. A calculational approach to flattening nested data parallelism on functional languages. In *Fifth International Workshop on Functional and Logic Programming, Marburg*, Jan 1996.

[9] Daniel W. Palmer, Jan F. Prins, and Stephen Westfold. Work-efficient nested data-parallelism. In *Prcoeedings of the Fifth Symposium in the Frontiers of Massively*, pages 186–193. IEEE, 1995.

[10] The Portland Group. *PGF77 User's Guide*, 1.1 edition, May 1994.

[11] Rice University. *High Performance Fortran Forum. High Performance Fortran Language Specification.*, 1.1 edition, Nov 1994.

[12] Thinking Machines Corporation. *C* Language Reference Manual*, 1991.