Generating Parallel Applications of Spatial Interaction Models

John Davy and Wissal Essah

School of Computer Studies, University of Leeds,
Leeds. West Yorkshire, UK, LS2 9JT

Abstract. We describe a tool enabling portable parallel applications of spatial interaction modelling to be produced automatically from high-level specifications of their parameters. The application generator can define a whole family of models, and produces programs which execute only slightly more slowly than corresponding hand-coded versions.

1 Introduction

This paper describes a prototype *Application Generator* (AG) for rapid development of parallel programs based on *Spatial Interaction Models* (SIMs). The work was a collaboration between the School of Computer Studies and the Centre for Computational Geography at the University of Leeds, with GMAP Ltd, a university company, as industrial partner. It was supported by the Engineering and Physical Sciences Research Council within its Portable Software Tools for Parallel Architectures programme.

An application generator can be characterised as a tool "with which applications can be built by specifying their parameters, usually in a domain-specific language" [1]. Our AG follows precisely this definition:

- both the SIM itself and the various computational tasks which use it are specified by defining parameters in a textual Interface File;
- parallel programs for the required tasks are generated from the Interface File, using templates which implement generic forms of these tasks;
- automatically generated programs deliver identical results to hand-coded versions of the same tasks.

Portability is achieved on two counts. Firstly, the level of abstraction of the Interface File is high enough to be machine-independent. Secondly, the AG generates source-portable programs in Fortran-77 with MPI. Currently the AG has been tested on a 4-processor SGI Power Challenge and a 512-processor Cray T3D. A serial version runs on a range of Sun and SGI workstations.

A common criticism of such high-level approaches to program generation is that excessive and unacceptable overheads are introduced in comparison with equivalent hand-coded programs. For this reason, every effort was made to ensure that needless inefficencies arising from genericity were avoided; also the AG determines an efficient form of the parallel program based on the dimensions

of the model and the number of processors available. With execution times for generated programs no more than 2% greater than the corresponding hand-coded programs, we have demonstrated that acceptable performance from high-level abstractions is indeed possible, at least within a limited application domain.

The work is closely related to *algorithmic skeletons* [3], in which common patterns of parallel algorithms are specified as higher-order functions, each of which has a template for parallel implementation. Our current prototype is only first order, but extensions to cover a wider range of industrial requirements are likely to lead to higher-order versions.

2 Spatial Interaction Modelling

Spatial interaction modelling was developed in the context of the social sciences, notably quantitative geography. A SIM is a set of non-linear equations which defines *flows* (people, commodities etc) between spatial zones. Such models are of importance to the business sector, academic researchers, and policy makers; they have been used in relation to a wide range of spatial interaction phenomena, such as movements of goods, money, services and persons, and spread of innovation.

Many realistic spatial interaction problems involve large data flows, together with computation which grows rapidly with the number of zones the model uses to represent geographical areas. Computer constraints have therefore limited the level of geographical detail that can practically be used for these models. Greater computing power, available by the exploitation of parallel processing, allows models to be used on a finer level with more realistic levels of detail and may enable better quality results. Despite these benefits, the exploitation of parallelism has received limited attention.

Recent research [2, 5, 6] has confirmed the reality of these benefits and the considerable potential of parallelism in this area. In particular, [5] shows the effectiveness of parallel genetic algorithms for solving the network optimisation problem, in comparison with previous heuristic methods. To date, however, such work has proceeded on an ad hoc basis, and there is still a need to provide a more uniform and consistent approach, enabling these technologies to be more readily exploited outside the specialised community of parallel processing.

The research reported here aims to rectify this deficiency by developing an application generator to:

- cover a wide range of SIM applications in the social sciences;
- enable efficient parallel implementations over a range of machines;
- be scalable to enable the use of large data sets on appropriate machines.

2.1 A Family of Models

There is a family of closely-related SIMs derived from so-called entropy-maximising methods [7]. A simple *origin-constrained* model is specified by

$$T_{ij} = O_i D_j A_i f(c_{ij}) \tag{1}$$

$$A_i = 1/\sum_j D_j f(c_{ij}) \tag{2}$$

where T_{ij} predicts the flow from origin zone i to destination zone j, O_i, D_j represent the 'sizes' of i and j, and c_{ij} is a measure of the travel cost from i to j (often simply the distance). The *deterrence function* $f(c_{ij})$ decreases as c_{ij} increases and is commonly modelled as either the exponential function $e^{-\beta c_{ij}}$ or the exponentiation function ${c_{ij}}^{-\beta}$ where β is an unknown *impedance factor* to be estimated. Equation (2) ensures that the predicted flows satisfy the constraint:

$$\sum_j T_{ij} = O_i \tag{3}$$

thus equating the 'size' of a zone to the number of flows starting there. A single *model evaluation* involves computing the set of values T_{ij} defined by (1) and (2).

Destination-constrained models are an obvious variant of origin-constrained models, in which (1) and (2) are replaced by

$$T_{ij} = O_i D_j A_j f(c_{ij}) \tag{4}$$

$$A_j = 1/\sum_i O_i f(c_{ij}) \tag{5}$$

thus satisfying the constraint

$$\sum_i T_{ij} = D_j \tag{6}$$

Doubly-constrained models are rather more complex:

$$T_{ij} = O_i D_j A_i B_j f(c_{ij}) \tag{7}$$

$$A_i = 1/\sum_j D_j B_j f(c_{ij}) \tag{8}$$

$$B_j = 1/\sum_i O_i A_i f(c_{ij}) \tag{9}$$

In this case the predicted flows satisfies constraints similar to both (3) and (6).

2.2 Applications Involving SIMs

SIMs are commonly used to represent human trip behaviour (such as to retail outlets) in network optimisation problems, where some 'profit' or other performance indicator of a global network is maximised. A typical problem is to locate some number N_F of facilities (e.g. shops, dealerships, hospitals) in a set of N_Z distinct zones (where $N_F < N_Z$) in such a way as to maximise the 'profit' from the facilities, which is computed from an underlying SIM by evaluating (a variant of) equation (1) for the zones in which facilities are located.

A necessary preliminary to solving this non-linear optimisation problem is the *calibration* of the model, in which the value of β is estimated from observed values T_{ij}^{obs} of trips (flows) between zones. This is a further non-linear optimisation problem which minimises an appropriate error norm $f(\beta)$ between T_{ij}^{obs} and the flows T_{ij} predicted by the model. Choice of appropriate error norms is a complex issue [7], which is beyond the scope of this paper; for simplicity we here use only maximum likelihood (10).

$$f(\beta) = \frac{\sum_i \sum_j T_{ij}^{\text{obs}} c_{ij} - \sum_i \sum_j T_{ij} c_{ij}}{\sum_i \sum_j T_{ij}} \tag{10}$$

The reliability of solutions to network optimisation problems depends on the reliability of the calibration process, which can be addressed computationally. The robustness of the computed value of β in relation to minor changes in T_{ij}^{obs} is assessed by *bootstrapping*, involving multiple recalibrations of the model, with slightly different sets of observed trip values obtained by systematic sampling and replacement from the original T_{ij}^{obs}. Thus the mean and variance of β can be derived. The number of calibrations of carried out, B is the *bootstrap size*. Clearly, greater values of B lead to more accurate estimates of the mean and variance of β; in practice, the heavy computation places practical limits on B, emphasising the potential benefits of parallel processing.

3 Parallel Implementation

We have implemented all three application components (calibration, bootstrapping, optimisation) on all three kinds of model (origin-, destination- and doubly-constrained), using Fortran77 with MPI. The aim of this part of the work was to experiment with alternative implementation techniques and derive templates for use in the AG. Details have already been reported in [4]; here there is space only to outline the principles in the origin-constrained case.

For all three components parallelism can be obtained within both the model evaluation and the application which evaluates the model: model evaluation is highly data parallel (as implied by (1) and (2)) and the applications can be parallelised by executing independent SIM evaluations in parallel.

For calibration we used a 'Golden Section' non-linear optimiser, which generates few independent model evaluations. Hence we expected parallelism to be exploited most effectively within the SIM evaluation. On the other hand, bootstrapping involves multiple independent calibrations and thus a high level of more coarsely-grained application-level parallelism was expected.

Following [5], a genetic algorithm was used to solve the non-linear network optimisation problem. Here the SIM is used to evaluate the fitness of each member of a population of possible solutions, hence multiple independent SIM evaluations are again involved, leading to plentiful application-level parallelism.

To explore the optimal combination of application- and model-based parallelism the P processors are viewed as a logical grid of dimensions $p_{\text{app}} \times p_{\text{mod}}$.

The application is parallelised between the p_{app} *application processors* of one row, each of which acts as a master processor for model evaluation, distributing its copy of the model between the p_{mod} *model processors* in its column.

3.1 Parallelism in the model

One of the main computational challenges for model evaluation is the large volume of stored data; space is required for the cost matrix c_{ij} and (for calibration and bootstrapping) the trip matrix T_{ij}^{obs}. A data parallel implementation distributes them across the processors using cyclic partitioning by rows: processor 1 receives rows 1, P+1, 2P+1,..., processor 2 rows 2, P+2, 2P+2,..., and so on. The matrices are treated slightly differently, as follows:

- each processor reads an (x, y) pair for each zone centroid, to compute its own rows of c_{ij}, stored in *dense* format.
- each processor stores the corresponding rows of T_{ij}^{obs} (also read from file), in *sparse* format since most trips are between physically close zones.

This distribution ensures that model evaluation is scalable from the perspective of memory usage, as long as the number of processors increases proportionately to the model size. Partitioning cyclically leads to a better load balance than partitioning contiguously because of the removal of systematic matrix patterns.

Evaluation of (1) and (2) may then proceed entirely in parallel with no further communication. It is only necessary to store one row of the T_{ij} matrix, since the error norms for calibration or bootstrapping are cumulatively evaluated in parallel from equation (10). The 'profit' in network optimisation is similarly accumulated.

3.2 Performance results

Performance results were obtained on a 512-processor Cray T3D machine, using 'journey-to-work' data derived from the 1991 UK census (see [6]). This records the numbers of journeys (T_{ij}^{obs}) out of and into all 10764 electoral wards. Costs c_{ij} are computed from distances between the centroids (x_i, y_i) of wards; in the case of intra-ward journeys the costs c_{ii} are fixed at some notional distance d_i depending on ward size. A second, smaller data set contains equivalent information aggregated into 459 electoral districts.

These data are representative of a range of other origin-constrained models, such as journeys to shopping centres or car dealerships, and can therefore be used to simulate corresponding location optimisation problems. As most journeys are between relatively close zones the trips matrix has the typical highly sparse pattern. Also, since the data sets satisfy both origin and destination constraints they may be used to assess doubly-constrained calibration and bootstrapping.

In all cases an exponential deterrence function was used. For space reasons, only a selection of the results in [4] are given.

As expected, calibration obtained best results with all parallelism in the model ($p_{\text{app}} = 1$), whereas bootstrapping and network optimisation enabled

Table 1. Optimisation times (sec) for locating 15 facilities in 10764 zones

	p_{app}				
P	1	2	4	8	16
32	3779	-	-	-	-
64	3317	1843	-	-	-
128	3300	1657	928	-	-
256	3459	1650	833	471	-
512	3736	1724	824	418	241

Table 2. Calibration times (sec) for 10764 zones

P	Time at $p_{app} = 1$	Speedup
32	274.3	-
64	137.6	1.99
128	69.2	3.96
256	35.1	7.8
512	17.9	15.3

maximum parallelism in the application. Table 1 shows optimisation times for allocating 15 car dealerships within 10764 zones, for varying values of P and p_{app}. Increasing p_{app} always decreases execution time, so the best time is obtained when the maximum value of p_{app} is used (in this case $P/32$, since the model requires 32 processors to evaluate). Similar results were obtained for the 459-zone model, which can be evaluated on a single processor (ie $P_{app} = P$). Encouragingly, the diagonal entries in Table 1 show a speedup close to linear.

Bootstrapping results show a similar pattern to network optimisation. Calibration was predictably less scalable, since only the finer-grain model parallelism was available. However, even here the results were encouraging. Table 2 shows a near-linear improvement in execution time as the number of processors increases 16-fold – the baseline of 32 processors was again the minimum necessary to evaluate the model. Even with the smaller 459-zone model useful performance improvements were obtained by parallelism up to 64 processors (see [4] for details) but performance degraded thereafter. By contrast, bootstrapping and optimisation continued to achieve near-linear speedup up to 512 processors even with the smaller model, because parallelism was exploited at the application level.

4 Specifying a Spatial Interaction Model

A SIM application can be specified by defining its parameters in a textual Interface File. We illustrate this for a simple case in Fig. 1, which defines a locational problem to optimise placement of 15 facilities within 10764 zones. (The need for MAXTRP in this file arises from using static arrays in the Fortran-77 implementation and is not inherent in the model specification.)

```
SIM                 % Model parameters
  MODTYP=0                % origin constrained model
  DETFUN=exp              % exponential deterrence function
CALIBRATE           % calibration parameters
  ERROR=maxlike           % maximum likelihood error norm
BOOTSTRAP           % bootstrapping parameters
  BSIZE=256               % size of bootstrap sample
OPTIMISE            % optimisation parameters
  LOC=15                   % number of locations to be optimised
DATASIZES           % data set details for
  NZ=10764                % number of origin zones
  MZ=10764                % number of destination zones
  MAXTRP=586645           % total number of non-zero trips
                          % in calibration data
```

Fig. 1. Interface File for simple origin-constrained model

This Interface File format could be used as the input to an AG to generate parallel programs using templates based on the codes described in section 3. In effect we would have a generator based on SIMs defined by equations (1) to (9). While interesting as a demonstration of principle, the simplified form of the models is very restrictive. To develop a more powerful AG, we studied the requirements of a wide range of spatial location problems, including examples from retailing, agricultural production, industrial location, and urban spatial structure. From these a more generic SIM formulation was derived. Here we outline the origin-constrained version, without seeking to justify the modelling process.

4.1 A Generic Origin-Constrained Model

We assume that each origin zone produces 'activities' or 'goods' of several types and that each destination zone has several facilities' (consumers of activities/goods). This rather general terminology describes a range of different phenomena; for instance (and rather counter-intuitively) 'goods' may be m different categories of potential buyers travelling to one of r different car dealerships.

The earlier flow value, T_{ij} generalises to T_{ij}^{mr}, the flow of good type m from zone i to facility r in zone j. Origin and destination sizes, O_i and D_j become O_i^m and D_j^r and we allow for the latter to be exponentiated. Thus (1), (2) and (3) generalise to

$$T_{ij}^{mr} = O_i^m (D_j^r)^\alpha A_i f(c_{ij}) \tag{11}$$

$$A_i = 1/\sum_j \sum_r (D_j^r)^\alpha f(c_{ij}) \tag{12}$$

$$\sum_j \sum_r T_{ij}^{mr} = O_i^m \tag{13}$$

The network optimisation problem is defined as maximising the difference between revenues (D_j^r) and costs (C_j^r) for each facility, ie for each r maximise

$$\sum_{j \in J} (D_j^r - C_j^r) \tag{14}$$

over a subset J of destination zones, subject to (13).

Revenues are computed as $\sum_i \sum_m p_i^m T_{ij}^{mr}$ where p_i^m is the revenue generated from a good of type m from zone i. In the most general case, costs at facility of type r at j have three components: maintaining the facility, transport, and the costs associated with transactions. This is modelled as

$$C_j^r = v_j^r (D_j^r)^\alpha + (\sum_i \sum_m c_{ij} T_{ij}^{mr}) + (q_j^r \sum_i \sum_m T_{ij}^{mr}) \tag{15}$$

where v_j^r, q_j^r are the unit costs of running facility r at j and making transactions there. The second and third terms are not always needed.

Finally, unit transport costs, c_{ij}, can be modelled by a term proportional to distance together with 'terminal costs', such as parking. Thus, in the most general case,

$$c_{ij} = t d_{ij} + \eta_i + \rho_j \tag{16}$$

where t is travel cost per unit distance, d_{ij} is a distance measure (we use Euclidean distance), η_i and ρ_j are the terminal costs at origin and destination. In some cases the t value may be irrelevant (ie it is only necessary to have costs proportional to distance) and the terminal costs are not always required.

Specifying this more general model requires the values of m, r and α, as well as stating whether the optional parts of the model are included. The values of ρ_j, η_i, v_j^r, q_j^r and p_i^m are data to be read from files at runtime if required.

Thus, the previous Interface File (Fig. 1) would be replaced by Fig. 2.

5 The Application Generator

The prototype AG implements the generic model above. It accepts an Interface File as input and generates the required parallel programs using templates based on the codes described in section 3. The user supplies an Interface File as above, or alternatively is prompted for parameters at a simple command-line interface. Each of the CALIBRATE, BOOTSTRAP, OPTIMISE sections is optional.

Using the results of section 3, calibration only exploits parallelism in the SIM evaluation, whereas bootstrapping and optimisation use maximum parallelism in the application, subject to limits imposed by the model size, derived from the NZ and MZ parameters. Thus, within the constraints of the program structure chosen, the values of p_{app} and p_{mod} give optimal execution time.

The use of a generic model can lead to superfluous and time-consuming computations, such as adding 0 when optional parts of the computation are not present, and needless exponentiation when ALPHA has its (common) value 1. The AG removes all such redundant computation, bringing execution times of

```
SIM
  MODTYP=0
  DETFUN=exp
  O_ACTIVITY=1  % number of activities at origins
  D_FACILITY=1  % number of facilities at destinations
  CFLAG=0       % travel costs NOT included in optimisation
  TFLAG=0       % terminal and unit travel costs NOT included
  PFLAG=0       % revenue costs NOT included in optimisation
  QFLAG=0       % transaction costs NOT included in optimisation
  ALPHA=1.0     % value of alpha
CALIBRATE
  ERROR=maxlike
BOOTSTRAP
  BSIZE=256
OPTIMISE
  LOC=7
DATASIZES
  NZ=10764
  MZ=10764
  MAXTRP=586645
```

Fig. 2. Interface file with generic SIM

generated code very close to hand-coded versions. Table 3 shows overheads of less than 2% on the Cray T3D for a variety of model types, application components and values of P (with 10764-zone model).

Source code for the AG is a mix of Fortran, Perl and Unix scripts. The templates consist of 2700 lines of code totalling some 97 kbytes, and the rest of the AG consists of 2500 lines (77 kbytes). It has been implemented and tested on an SGI Power Challenge, a Cray T3D, and various Sun and SGI workstations.

6 Evaluation and Conclusions

The prototype AG shows that it is possible to generate non-trivial parallel applications from very high-level specifications, with small overheads compared with direct coding. This is made possible by a restricted application domain, which nevertheless is useful in practice and benefits from parallelism.

The current tool is not yet of industrial-strength, on several counts. It has not been adequately evaluated by real users, does not cover the full range of spatial interaction modelling variants, and is limited in the outputs produced. Further generalisation will require more detailed inputs from domain experts, including a standardisation of data formats in a flexible way to permit greater generality. More control over the optimisation process (such as specifying the number of generations for the genetic algorithm) is desirable. Lastly, the user interface betrays the system's Fortran origins and would benefit from a graphical presentation.

Table 3. Overheads of application generator

application type	model constraint	P	overheads (%)
calibration	origin	1	1
		2	1.26
		8	1.4
calibration	double	1	1.8
		2	1.7
		8	1.6
optimisation	origin	1	1.6
		4	1.85
		16	1.9

Despite its limitations, we believe this work is valuable in showing that rapid parallel application development through a skeleton-like approach can be made to work effectively in an appropriate, well-defined application domain.

References

1. C. Barnes and C. Wadsworth. Portable software tools for parallel architectures. In *Proceedings of PPECC'95*, 1995.
2. M. Birkin, M. Clarke, and F. George. The use of parallel computers to solve non-linear spatial optimisation problems. *Environment and Planning A*, 17:1049–1068, 1994.
3. M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation.* Pitman/MIT Press, 1989.
4. W. Essah, J. R. Davy, and S. Openshaw. Systematic exploitation of parallelism in spatial interaction models. In *Proceedings of PDPTA '97*, July 1997.
5. F. George. Hybrid genetic algorithms with immunisation to optimise networks of car dealerships. Edinburgh Parallel Computing Centre, EPCC–PAR–GMAP, 1994.
6. I. Turton and S. Openshaw. Parallel spatial interaction models. *Geographical and Environmental Modelling*, 1:179–197, 1997.
7. A. G. Wilson. A family of spatial interaction models, and associated developments. *Environment and Planning*, 3:1–32, 1971.