Un Modelo de Programación Paralelo para Memoria Distribuida

Jacqueline Fernandez, Mónica Fuentes, Fabiana Piccoli, Marcela Printista *
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950
5700 - San Luis
Argentina
e-mail: {gfuentes, jmfer, mpiccoli, mprinti}@unsl.edu.ar

1 Introduction

En los primeros estados de la computación paralela una de las soluciones más comunes para introducir paralelismo fue extender un lenguaje secuencial con algún conjunto de *constructores* comunmente denominados *forall*. Aunque la mayoría tiene la misma sintaxis, ellos difieren en su semántica e implementación. Entre las implementaciones más populares están las propuestas por High Performance Fortran (HPF)[7] y OpenMP [9]. En este trabajo se propone una extensión a la iteración *forall* para el lenguaje *C*. La iteración *forall* define un modelo de programación, el cual se caracteriza por:

- Permitir y explotar varios niveles de paralelismo: *Paralelismo anidado*[2], [3].
- Expandir el modelo de pasaje de mensajes.
- Posibilitar la asociación de un modelo de complejidad.

En consecuencia, este modelo brinda, por un lado la posibilidad de anidar sentencias *forall* y generalizar el modelo de programación de pasaje de mensajes (cada *forall* crea un comunicador, hablando en términos de MPI[8]); por el otro, posibilita el análisis y la predicción de la performance en base al modelo de complejidad asociado.

2 Modelo: Sintaxis y Semántica

Como se mencionó, el modelo de programación que se propone extiende el paradigma secuencial imperativo con la inclusión de nuevas sentencias: Iteraciones Paralelas. A continuación se muestra una versión simplificada de su sintaxis:

forall(i,first,last,f(i),res$_i$,size$_i$)

La variable i representa la variable de iteración, la cual toma valores enteros entre $first$ y $last$. Cada iteración sobre i es independiente de las demás y se realiza en paralelo. Las variables res_i y $size_i$ representan, respectivamente, el resultado de la i-th iteración y su correspondiente tamaño.

*Grupo subvencionado por la UNSL y ANPCYT (Agencia Nacional para la Promoción de la Ciencia y Tecnología)

Para explicar la semántica se necesita definir una máquina paralela. Supongamos una arquitectura con infinitos procesadores, cada uno con su memoria privada y unidos a los otros a través de una red de interconección. Los procesadores son organizados en grupos, al comienzo de la computación todos pertenecen al mismo grupo. Los procesadores de un mismo grupo tienen el mismo estado de memoria, y se asume que tienen los mismos datos de entrada y ejecutan el mismo programa. Cada procesador es una máquina $RAM[1]$ y se diferencia de las demás en el estado de su registro interno, inaccesible para el programador, el cual contiene su nombre, *NAME*. Cuando un procesador encuentra una iteración *forall* decide, según su *NAME*, el valor de i y en consecuencia a que subgrupo pertenece y cual es su nuevo $NAME$. Los procesadores de cada subgrupo ejecutan la correspondiente $f(i)$. Las siguientes expresiones muestran como se realiza la división de los procesadores, el mapping de estos a las distintas tareas y el intercambio de los resultados.

Cantidad de Iteraciones a realizar: $M = last - first + 1$
Iteración a hacer: $i = first + NAME \,\%\, M$
for (j = 1; j < M; j++)
$partner[j] = \Phi + (NAME + j) \,\%\, M$
donde Φ es: $\Phi = M \times (NAME \,/\, M)$
El nuevo valor de NAME es calculado por:$NAME = NAME \,/\, M$

Al finalizar el *forall*, cada procesador recupera su propio valor de *NAME*. Cada vez que una iteración *forall* es ejecutada, la memoria de los procesadores del grupo contiene exactamente los mismos valores. En tal punto la memoria es dividida en dos partes: aquella que será modificada y aquella que no tendrá ningún cambio dentro de la iteración. Las variables que pertenecen al segundo conjunto estarán disponibles en el ámbito del *forall* para lectura. Las otras serán particionadas entre los distintos grupos. En todo momento de la ejecución, los procesadores pertenecientes a un mismo grupo deben tener una visión consistente de la memoria y parecer un mismo proceso. Para lograr esto es necesario proveer de un mecanismo de intercambio de información entre los procesadores socios dentro del *forall*.

La semántica impone dos restricciones:

1. Los resultados res_i y res_j correspondientes a la ejecución de las iteraciones independientes i y j, cumplen con:

$$res_i \bigcap res_j = \emptyset \ \forall i, j$$

2. Para todo T_i (denominamos T_i a la ejecución del cuerpo de la i-th iteración $f(i)$), el espacio de memoria para sus resultados debe ser ubicado previamente a la ejecución de T_i. Esto indica que no es posible usar estructuras dinámicas de memoria, como listas y árboles, en el resultado de cada iteración.

La primer condición es mandatoria, durante la ejecución de cada T_i, los resultados res_i no pueden compartir direcciones de memoria.

3 Mapping y scheduling

Desafortunadamente una máquina infinita, tal como la descripta aquí, es una máquina ideal. Las máquinas reales poseen un número limitado de procesadores. Cada procesador tiene un registro interno, *NUMPROCESSORS*, donde se almacena el número de procesadores disponibles en su grupo corriente.

Tres diferentes situaciones deben ser consideradas en la ejecución de un *forall* con $M = last - first + 1$ iteraciones:

- $NUMPROCESSORS = 1$;
- $M \geq NUMPROCESSORS$;
- $NUMPROCESSORS > M$;

El primer caso es trivial. Un único procesador ejecuta todas las iteraciones secuencialmente y no existe la oportunidad de explotar el paralelismo intrínseco.

El segundo caso ha sido ampliamente estudiado como paralelismo chato. Su principal problema es el balance de carga. Para solucionarlo varias políticas de scheduling fueron propuestas [9]. Muchas políticas de asignación son posibles, la asignación puede ser por: *bloques*, *bloques-cíclica*, *guiada* o *dinámica*.

El tercer caso trae consigo problemas adicionales como es el balance de la carga. Su solución implica que los subgrupos en que se dividieron los procesadores no serán del mismo tamaño. La solución al balance de la carga se encuentra en la modificación de la política de distribución de procesadores vista anteriormente. Solucionar el balance de la carga trae consigo otro problema: determinar la relación de sociedad entre los procesadores de distintos grupos. La relación de sociedad genera un *polytope* donde cada procesador, en cada grupo, tiene la propiedad de tener uno y sólo un socio, al cual le envía sus resultados.

Actualmente, existe una implementación del modelo, la cual utiliza la librería de pasaje de mensajes sobre *puBSP*[4]. Considerando el modelo y como un test exahustivo, se están desarrollando varias aplicaciones. Entre ellas se encuentran: la *Multipicación de Matrices*[10], *La Transformada rápida de Fourier*[6] y el problema de los *N-cuerpos*[5].

4 Concluiones

El modelo de programación aquí expuesto, permite expresar paralelismo a través de las sentencias *forall*. Admite además, varias extensiones como son las claúsulas de reducción.

El modelo propuesto extiende el modelo de pasaje de mensajes estandar, a la vez que se caracteriza por ofrecer:

- Facilidad de Programación.
- Portabilidad a otras plataformas.
- Predictibilidad de la performance al tener asociado un modelo de predicción confiable.

Todas estas características lo hacen un modelo muy interesante para la investigación y el desarrollo de otros modelos orientados a él.

Referencias

[1] Aho, A. V. Hopcroft J. E. and Ullman J. D.: The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts, (1974).

[2] Ayguade E., Martorell X., Labarta J., Gonzalez M. and Navarro N. Exploiting Multiple Levels of Parallelism in OpenMP: A Case Study Proc. of the 1999 International Conference on Parallel Processing, Aizu (Japan), September 1999.

[3] Blikberg R., Sørevik T.. Nested parallelism: Allocation of processors to tasks and OpenMP implementation. Proceedings of The Second European Workshop on OpenMP (EWOMP 2000). Edinburgh, Scotland, UK. 2000

[4] Bonorden O., Huppelshauser N., Juurlink B., Rieping I. PUB library, Release 6.0 - User guide and function reference. University of Paderbon, Germany. (1998)

[5] Barnes J., Hut P. A hierarchical O(NlogN) force calculation algorithm. nature. Vol 324.P. 446,1986.

[6] Cooley, J. W. and Tukey, J. W.: An algorithm for the machine calculation of complex Fourier series, Mathematics of Computation, **19**, 90, (1965) 297–301.

[7] High Performance Fortran Forum: High Performance Fortran Language Specification. Version 2.0 http://dacnet.rice.edu/Depts/CRPC/HPFF/ versions/hpf2/hpf-v20/index.html (1997)

[8] MPI Forum: MPI-2: Extensions to the Message-Passing Interface, http://www.mpi-forum.org/docs/mpi-20.ps.Z (1997).

[9] OpenMP Architecture Review Board: OpenMP Specifications: FORTRAN 2.0. http://www.openmp.org/specs/ mp-documents/fspec20.ps (2000).

[10] Quinn M. Parallel Computing. Theory and Practice. Second Edition. McGraw-Hill, Inc.