PARADIGMAS DE PROGRAMACION PARALELA

Saez Fernando, Piccoli Fabiana, Printista Marcela, Raúl Gallard
L.I.D.I.C. *
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950,
(5700) San Luis, Argentina
mprinti@unsl.edu.ar

1 Introducción

En programación, la palabra paradigma es a menudo usada con una connotación general como "metodologías de alto nivel que se reconocen en muchos de nuestros algoritmos efectivos". En este trabajo se usará el término en un sentido más preciso: *un paradigma de programación es una clase de algoritmos que resuelven problemas diferentes pero tienen la misma estructura de control.* Estos programas a menudo son llamados *esqueletos algorítmicos o programas genéricos* [2]. A partir de estos esqueletos se pueden derivar *programas modelos* [1] que ilustren como el paradigma resuelve problemas específicos.

Un esqueleto incluye tipos de datos no especificados y procedimientos que varían de una aplicación a otra. Un programa modelo luego se obtiene reemplazando estos tipos de datos y procedimientos con los tipos de datos correspondientes y con los procedimientos que constituyen el programa secuencial que resuelve el problema específico. La esencia de esta metodología de programación es que el programa modelo tiene una componente paralela que implementa un paradigma y una componente secuencial específica de una aplicación.

Los temas presentados en este trabajo forman un área de interés dentro de la línea de investigación en Paralelismo. A diferencia de lo que ocurre en otros trabajos de la misma línea, el énfasis en este trabajo no está centrado sólo en aspectos de performance, sino en tratar de distinguir los aspectos de programación que son propios de un paradigma de los detalles propios de una aplicación específica. Esto es esencial, para escribir programas paralelos que sean fáciles de entender.

2 Divide y Vencerás

El paradigma `Divide and Conquer` ($D\&V$) [3, 6] es un método elegante de resolver problemas. Simplemente se divide el problema en problemas mas pequeños de la misma clase, los que se resuelven separadamente y cuyos resultados parciales se combinan para obtener la solución final. Este método es utilizado recursivamente para dividir las partes en problemas mas y mas pequeños hasta alcanzar un punto donde el problema se resuelve trivialmente.

*Grupo subvencionado por la UNSL y ANPCyT (Agencia Nacional para Promover la Ciencia y Tecnología)

```
1 parD&V(tabla, first, last)
2 {
3   if first<last then
4    {
5        divide(tabla, first, last, middle)
6        PARALLEL(parD&V(tabla, first, middle),
7                 parD&V(tabla, middle+1, last))
8        combine(tabla, first, last, middle)
9
10   }
11   else soluciontrivial(tabla)
12 }
```

Figura 1: Esqueleto $D\&V$ paralelo

2.1 Implementación del Paradigma

Para la implementación de este paradigma se comenzó analizando el paradigma secuencial $D\&V$. Este cumple 3 propiedades simples:

- Para un problema de tamaño n, su solución queda definida para un arreglo de n elementos.
- Para un problema de tamaño 1, éste mismo es su propia solución.
- Para un problema de tamaño 1, su solución se obtiene dividiendo el problema en mitades, las cuales pueden ser resueltas separadamente.

Luego de establecer estas propiedades y de analizar su propiedad recursiva inherente se detectó que la arquitectura de programa que mejor se adapta a este tipo de paradigma es un *árbol binario de procesadores*. Al iniciar el proceso, existe un único procesador que actúa como nodo raíz y que inicia con el problema completo. Este procesador, denominado procesador *padre*, genera la división del problema y realiza la distribución de los subproblemas a otros dos procesadores, denominados *hijos*. Cada uno de estos hijos se convierte en la raíz de un nuevo sub-árbol y el proceso es repetido (Fase de división). Cuando el problema ya no es mas divisible comienza el proceso de combinación. Cada procesador hijo toma su solución trivial y se la envía al padre, quien luego de combinar resultados repite el procedimiento (Fase de conquista). Finalmente el procesador principal (raiz) obtiene la solución del problema completo.
La Figura 1 muestra el pseudocódigo del esqueleto del paradigma paralelo.
Como se puede observar en la Figura 1, las invocaciones recursivas a los procedimientos `parD&V` se realizan dentro de un bloque estructurado que implementa el paralelismo. Específicamente el constructor `PARALLEL` implementa la distribución de tareas a los dos nuevos procesadores *hijos*. En la implementación de `PARALLEL` no fue sólo necesario asignar a los procesadores hijos subtareas sino también realizar la distribución de datos entre los nuevos hijos, mantener la pista de procesos hermanos y la sincronización de los procesadores.
Para este paradigma el programa modelo escogido fue la *Fast Fourier Transform* [3, 4, 5, 6]. Este programa computa la frecuencia de las componentes de una señal que han sido observada n veces. Este resultó ser un buen caso de test ya que en esta aplicación estuvieron involucrados todos los procedimientos mencionados en el esqueleto de $D\&V$ en su versión paralela. Esto no fué así en otras aplicaciones que se escogieron como candidatas, como fué el caso del algoritmo del `quicksort`. En este caso el algortimo consistía sólo de la fase de división ya que al llegar al problema trivial la solución al problema queda distribuida entre los procesadores hojas del árbol.

```
P(a1,b1)
Q(a2,b1)     P(a2,b2)
Q(a3,b1)     Q(a3,b2)       P(a3,b3)
Q(a4,b1)     Q(a4,b2)       Q(a4,b3)
.......       ......         ......       .....    P(an-1,bn-1)
Q(an,b1)     Q(an,b2)       Q(an,b3)     .....    Q(an,bn-1)
```

Figura 2: Matriz de precedencia All-Pairs

3 All Pairs Pipeline

Un problema `All-Pairs` es una computación sobre cada posible subconjunto que consiste de dos elementos escogidos de un conjunto de n elementos. Definimos el problema concisamente por medio de Matrices de Precedencias y luego derivamos en un algoritmo paralelo.
En este punto desarrollamos el paradigma `All-Pairs` discutido por Cosnard 1988 [2].

3.1 El Problema All-Pairs

Sea A un conjunto de n elementos:

$$A = a1, a2, , ..., an$$

Existen $(n-1) * n/2$ maneras de seleccionar un subconjunto de A que consiste de dos elementos:

```
{a2,a1}
{a3,a1}   {a3,a2}
{a4,a1}   {a4,a2}   {a4,a3}
 ....       ....               .....
{an,a1}   {an,a2}   {an,a3}   ..... {an,an-1}
```

Cada subconjunto a_i, a_j puede ser representado por un par ordenado (a_i, a_j) donde a_i y a_j son elementos de A, y $1 <= j <= i <= n$. Una computación `All-Pairs` realiza una operación $Q(a_i, a_j)$ sobre cada par (a_i, a_j). Esta operación transforma a_i y a_j sin involucrar ningún otro elemento de A.
Los elementos de la matriz de precedencia son operaciones. En otras palabras $Q(a_i, a_j)$ es precedido por $Q(a_{i-1}, a_j)$ y $Q(a_i, a_{j-1})$ y es seguido por $Q(a_{i+1}, a_j)$ y $Q(a_i, a_{j+1})$.
Encontramos en la implementación que era conveniente trabajar con dos estructuras; la estructura original a y otra estructura temporaria b, donde realizar las computaciones. La Figura 2 muestra la matriz de precedencia para la variante propuesta del `All-Pairs`. Sobre cada par de elementos se definen dos operaciones:

- La operación $P(a_i, b_i)$ transforma el elemento a_i y computa el correspondiente elemento b_i, donde $1 <= i <= n-1$.
- La operación $Q(a_i, b_j)$ transforma elemento a_i y b_j, donde $1 <= j <= i <= n$.

```
1 pipeAllPairs(a,b,1,n)
2  {
3     for (i=r to s) {
4        receive(partnerleft, a[i])
5        for (j=r to i-1)
6             Q(a[i], b[j])
7        P(a[i], b[i])
8     }
9     for (j=s+1 to n) {
10      receive(partnerleft, a[column.j])
11      for (i=r to s)
12           Q(a[column.j], b[i])
13      send(partnerright, a[column.j])
14    }
15    for (i=s down r)
16         send(partnerright, a[i])
17       for (j=r-1 down 1) {
18            receive(partnerleft, a[column.j])
19            send(partnerright, a[column.j])
20       }
21 }
```

Figura 3: Esqueleto Paralelo All-Pairs Pipeline

3.2 Paradigma Paralelo

En la técnica de `Pipeline` el problema es dividido en una serie de tareas que tienen que ser completadas una después de la otra. En efecto ésto es la base de programación secuencial. En un pipeline cada tarea será ejecutada en un procesador separado. Cada etapa contribuye al problema entero y transmite información que es necesaria para etapas siguientes. Dado que el problema puede ser dividido en una serie de tareas secuenciales como el pipeline, puede proveer mayor aceleración bajo los siguientes tres tipos de computaciones.

- Si mas de una instancia del problema completo es ejecutado.
- Si una serie de datos debe ser procesado, y cada uno requiere de múltiples operaciones.
- Si la información para comenzar el siguiente proceso puede ser pasada antes que el proceso haya completado todas sus operaciones internas.

Si el número de etapas es mayor que el número de procesadores en cualquiera de los tres casos, un grupo de tareas puede ser asignado a cada procesador.
En la Figura 3 se muestra el pseudocódigo del esqueleto del paradigma paralelo. Se puede observar que el esqueleto atraviesa por cuatro fases:

- Fase de Entrada: El nodo entra elementos a_r hasta a_s y los almacena en un arreglo local a. Cada elemento entrante a_i interactúa con cada uno de los previamente almacenados a_r hasta a_{i-1}.
- Fase de Transferencia: El nodo entra los elementos a_{s+1} hasta a_n. Cada elemento a_j interactúa con cada elemento local y es inmediatamente sacado al próximo nodo.
- Fase de salida: El nodo saca los elementos locales en orden reverso.

- Fase de Copia: El nodo copia todos los elementos sacados por el nodo previo en orden reverso.

El Paradigma `All-Pairs` es ilustrado por un pipeline que resuelve una *Householder Reduction* [2] de una matriz a una forma triangular. Esta aplicación resuelve un sistema de n-ecuaciones lineales reduciendo este sistema a una matriz de forma triangular tal que puede ser resuelta fácilmente por sustitución hacia atras.
Para una computadora paralela, la *Reducción de Householder* es un método atractivo porque es numéricamente estable y no requiere el cálculo de un *pivote* [2]. En este punto derivamos un algoritmo pipeline para esta aplicación directamente a partir del esqueleto del paradigma `All-Pairs`. Es importante destacar que para el programa modelo seleccionado, tal como se obseva en el esqueleto algorítmico, fué necesario sólo instanciar el código de un nodo del esqueleto al proceso encargado de atender las columnas r hasta s, donde $1 <= r <= s <= n-1$. El pipeline ingresa las columnas en orden natural, reduce la matriz a una forma triangular y entrega las columnas finales en orden reverso.

4 Conclusiones

La explosión exitosa de las computadoras paralelas depende de desarrollos de conceptos útiles en el cual los programadores opten por ver diferentes aplicaciones como variaciones de un tema común. Los conceptos mas fundamentales tales como procesos paralelos y comunicación de mensajes, son embebidos en lenguajes de programación. En otros casos descubrimos paradigmas de programación que pueden ser usados para resolver cierta clase de aplicaciones. En esto último radica la principal motivación de la realización de este trabajo.
Escribir *esqueletos algorítmicos* nos condujo a clarificar muchas soluciones paralelas para una variedad de problemas que comparten las mismas estructuras de control paralela. En particular, además de proveer una buena performance, nos habilit'o a una programación eficiente y nos permiti'o enfocarnos sobre las subpartes secuenciales específicas y no sobre la estructura del programa paralelo.

Referencias

[1] Foster I. *Designing and Building Parallel Programs*, (Addison-Wesley,1994).
[2] Hansen P.B. *Brinch Hansen on pascal Compilars* - prentice Hall. 1985
[3] Gibbons A. Rytter W. *Efficient Parallel Algorithms*, (Cambridge University Press, 1988)
[4] Rodríguez Leon C., *El diseño y Análisis de Algoritmos Paralelos*, (Colección textos universitarios, ISBN: 84-8416-220-6, Cabildo de Canarias. Conserjería de Educación)
[5] Quinn M.J. and Hatcher P. *Data Parallel Programming on Multicomputers* (IEEE Software, Vol. 7 1990).
[6] Wilkinson B. and Allen M. *Parallel Programming: Techniques and Application using Networked Workstations and Parallel Computers* (Prentice-Hall, 1999).