Conclusiones

Sabedores de la importancia de la computación dispersa e irregular, hemos profundizado en esta tesis en la paralelización de este tipo de problemas. Hemos elegido la resolución de ecuaciones dispersas como caso de estudio por constituir un problema con muchos de los paradigmas representativos de la computación irregular. En resumen, en nuestra búsqueda hemos programado en C, F90 y F77, para plataformas como el CRAY T3D, T3E y el SGI Power Challenge, hemos utilizado librerías de comunicación PVM, MPI y SHMEM, los entornos de paralelismo de datos CRAFT y HPF-2, hemos extendido la librería DDLY y usado la herramienta Cocktail en el desarrollo del compilador, hemos comparado con CHAOS, y finalmente hemos identificado algunas limitaciones de los paralelizadores PFA y Polaris, proponiendo soluciones.

Las principales aportaciones y las líneas de investigación que pretendemos seguir en el futuro se discuten a continuación.

Principales aportaciones

Volviendo a la analogía, comentada en el prefacio, entre la investigación y la búsqueda del camino apropiado en una región desconocida, en esta tesis hemos topografiado tres rutas para alcanzar soluciones al problema de la resolución paralela de sistemas de ecuaciones dispersos: la paralelización manual, semi-automática y automática.

Paralelización manual

- Hemos desarrollado un algoritmo secuencial, SpLU, para resolver sistemas de ecuaciones dispersos, que incluye las etapas de análisis-factorización y resolución triangular. El algoritmo secuencial conduce en muchos casos a un error de factorización y a un llenado menor que el que consigue la rutina estándar MA48. En general, en los experimentos realizados nuestro algoritmo SpLU ha resultado ser más estable que la rutina MA48. Adicionalmente el tiempo de factorización puede llegar a ser bastante menor que el de la MA48.

- Hemos comprobado que la rutina MA48 no puede ser paralelizada eficientemente debido a su organización *left-looking*. Sin embargo nuestro algoritmo exhibe un alto grado de paralelismo, no sólo por basarse en una aproximación *right-looking*, sino también por explotar la dispersión de la matriz agrupando un cierto número de pivots compatibles. El algoritmo paralelo presenta una eficiencia muy alta, superior a las que proporcionan otros trabajos previos en la misma línea.

Paralelización semi-automática

- Hemos identificado las limitaciones de los compiladores de paralelismo de datos CRAFT y HPF a la hora de expresar códigos dispersos. Básicamente estas limitaciones se derivan de la imposibilidad de estas herramientas para manejar estructuras de datos dinámicas o comprimidas. Esto impide la paralelización eficiente de algoritmos dispersos dado que se apoyan en las mencionadas estructuras de datos. Nuestra aportación ha consistido en extender la capacidad de HPF-2, mediante la inclusión de una directiva (`SPARSE`) y la extensión semantica de otras ya existentes. De esta forma, permitimos al programador escribir códigos en F90 basados en estructuras de datos con listas enlazadas o vectores comprimidos, delegando en el compilador la tarea de generar la versión paralela. La idea más importante de nuestra extensión es permitir al programador especificar mediante directivas, no sólo la distribución de los datos, sino también la estructura de estos datos en memoria. Las extensiones son mínimas y simples pero al mismo tiempo suficientes para proporcionar una gran flexibilidad al programador.

- Adicionalmente, hemos desarrollado parte del módulo de compilación que completa el compilador de HPF-2 para soportar las nuevas extensiones propuestas. Más precisamente, disponemos del analizador léxico y sintáctico para reconocer la directiva añadida, junto con las rutinas en tiempo de ejecución encargadas de la distribución de datos y comunicaciones. Para construir los analizadores hemos hecho uso de la herramienta Cocktail para diseño de compiladores, mientras que las rutinas en tiempo de ejecución se han construido siguiendo la filosofía de las DDLY.

- Hemos simulado la salida paralela generada por nuestro compilador, partiendo de un algoritmo secuencial en F90 anotado con las nuevas directivas, comparable en prestaciones a la rutina MA48. La salida paralela ha resultado ser bastante eficiente, presentando el mismo error de factorización y llenado que la versión secuencial. También hemos comparado nuestra aproximación con la técnica Inspector-Executor implementada en la librería CHAOS para la paralelización de códigos irregulares, resultando esta última claramente insuficiente.

Paralelización automática

- Hemos analizado la salida de dos paralelizadores automáticos, PFA y Polaris, cuando la entrada es un código de factorización dispersa en F77. Ninguno de los dos obtiene buenos resultados debido a ciertas limitaciones en las técnicas de detección y paralelización de ambos compiladores. Para resolverlo hemos introducido cuatro técnicas para la detección de dependencias y paralelización automática de códigos dispersos: monotonía en los arrays que acotan bucles, no solapamiento de variables de inducción, detección de vectores de permutación, y extensión de las técnicas de privatización.

- Hemos aplicado estas técnicas a nuestro algoritmo de factorización en F77 obteniendo eficiencias muy altas en la SGI Power Challenge de memoria compartida. Adicionalmente estas técnicas se han mostrado válidas en otros códigos irregulares dispersos, justificándose así su necesaria implementación en los paralelizadores automáticos.

Líneas de investigación futura

Nuestro algoritmo SpLU para la resolución paralela de sistemas de ecuaciones dispersos puede ser mejorado en dos puntos principales. El primero está orientado a reducir el coste de las comunicaciones aplicando una distribución cíclica por bloques en lugar de una cíclica pura. El segundo se centra en reducir el movimiento de datos y facilitar el insertado de entradas utilizando una estructura de listas enlazadas desordenadas tanto por filas como por columnas. Estos dos aspectos unidos a un mayor cuidado en el aprovechamiento de la cache pueden resultar en mejores prestaciones.

Sin embargo, más que optimizar un problema particular, nos parece más interesante simplificar las técnicas de programación paralela ya sea de forma semi-automática o automática. Nosotros hemos abordado la construcción de un módulo de compilación que procesa sólo la directiva `SPARSE` introducida, pero también es necesario contemplar extesiones para problemas no estructurados. En cuanto a la aproximación automática, las técnicas propuestas para poder paralelizar códigos basados en estructuras de datos comprimidas, ya están siendo implementadas en Polaris en colaboración con el grupo de investigación dirigido por el profesor David Padua en Illinois.

En la misma línea de la paralelización automática, aparte de las estructuras de datos comprimidas, es necesario trabajar en otras situaciones con estructuras de datos dinámicas basadas en punteros, como las listas, árboles, etc. La detección automática de paralelismo de este tipo de problemas presenta un gran desafío al que nos queremos enfrentar. Por ejemplo, los códigos de factorización LU basados en listas enlazadas en C o en F90 no pueden ser paralelizados automáticamente con la tecnología actual.

Dando un paso más en complejidad y en aras de facilitar la tarea al programador, no descartamos la idea de dedicar algunos esfuerzos al campo de los reestructuradores de códigos densos secuenciales en dispersos y paralelos. Bik y Pingali presentan algunas soluciones para generar un código disperso secuencial a partir del código denso asociado, pero por el momento, sus resultados aún dejan un amplio margen para la investigación en este campo. La ventaja que presenta abordar la paralelización automática de un código disperso partiendo de la versión densa, estriba principalmente en la mayor cantidad de información que puede obtener el compilador. Más precisamente, el análisis de dependencias de algoritmos numéricos dispersos se simplifica si partimos de la versión densa.