Descripción de primitivas de BSPlib

Inicialización

Nombre:
bsp_begin : Produce un número de procesos de BSP.

En C:

```
#include "bsp.h"
void bsp_begin(int maxprocs);
```

Descripción:
La mayoría de los procesos maxprocs son creados en un programa BSPLib por las operaciones **bsp_begin** y **bsp_end**. Clasifican una sección del código que se ejecutará de una manera SPMD en un número de procesadores. Puede solamente haber una ocurrencia de un par de **bsp_begin/bsp_end** dentro de un programa, aunque hay dos diversas maneras de comenzar un programa bsplib:

1. Si **bsp_begin** y **bsp_end** son la primera y última declaración en un programa, entonces el cómputo BSPLib es toalmente SPMD.
2. Un modo alternativo está disponible donde un solo proceso comienza la ejecución y determina el número de procesos paralelos requeridos para el cálculo. Puede entonces generar el número requerido de procesos usando **bsp_begin**. La ejecución de los procesos generados entonces continúa de un modo SPMD, hasta que **bsp_end** es encontrado por todos los procesos. En ese punto, se termina todo menos el proceso cero, el que se deja para continuar ejecución el resto del programa secuencialmente.

Nombre:
bsp_end : Termina los procesos de BSP.
En C:

```
#include "bsp.h"
void bsp_end();
```

Descripción:
Termina la ejecución de SPMD de los procesos generados por **bsp_begin**. Todo se termina menos el proceso cero, el cual se deja para continuar la ejecución del resto del programa secuencialmente.
Nombre:
bsp_init : Inicializa el sistema BSPLib.
En C:

```
#include "bsp.h"
void bsp_init(void (*startproc)(void), int argc, char **argv);
```

Descripción:
El propósito de **bsp_init** es apoyar un modo de inicialización de proceso que permita a un solo proceso empezar la ejecución y determinar número de procesos paralelos requeridos para un cálculo. El número deseado de procesos se genera entonces usando **bsp_begin**.

Alto

Nombre:
bsp_abort : Detiene un cómputo BSP.
En C:

```
#include "bsp.h"
void bsp_abort(char *format,...);
```

Descripción:
La función **bsp_abort** se puede utilizar para imprimir un mensaje de error seguido por un alto del programa BSPLib. La rutina se diseña para no requerir una sincronización de barrera de todos los procesos.

Pregunta

Nombre:
bsp_nprocs : Determina el número total de procesos BSP.
En C:

```
#include "bsp.h"
int bsp_nprocs();
```

Descripción:
Si la función **bsp_nprocs** se llama antes de **bsp_begin**, entonces devuelve el total de los procesadores que son accesibles en línea. Si se llama después de **bsp_begin** devuelve *n*, el número real de procesos distribuidos en el programa, donde 1 <= *n* <= *maxprocs*, y *maxprocs* es el número de procesos solicitados en **bsp_begin**.
Nombre:
bsp_pid : Determina el identificador de proceso de un proceso BSP.
En C:

```
#include "bsp.h"
int bsp_pid();
```

Descripción:
La función **bsp_pid** devuelve el identificador de proceso del proceso que ejecuta la llamada de la función; donde 0 <= bsp_pid() < bsp_nprocs()

Nombre:
bsp_time : Reloj de tiempo real de alta precisión.
En C:

```
#include "bsp.h"
double bsp_time();
```

Descripción:
La función **bsp_time** da acceso a un reloj de alta precisión. Esta función es una operación local de cada proceso, y puede ir en cualquier punto después de **bsp_begin**. El resultado del temporizador es el tiempo en segundos desde **bsp_begin**.

Superstep

Nombre:
bsp_sync : Final de un superstep.
En C:

```
#include "bsp.h"
void bsp_sync();
```

Descripción:
Un cálculo BSPLib consiste en una secuencia de supersteps. Durante un superstep cada proceso puede realizar un número de cómputos sobre datos contenido localmente al principio del superstep y puede comunicar datos a otro proceso utilizando **bsp_put**, **bsp_get**, **bsp_hpput**, **bsp_hpget**, o **bsp_send**. Cualquier comunicación dentro de un superstep está garantizada para ocurrir para el final del superstep, donde **bsp_nprocs** procesos sincronizan en una barrera. El fin de un superstep y el principio siguiente se identifican por una llamada al procedimiento biblioteca **bsp_sync**. No se garantiza que ocurra la

comunicación iniciada durante un superstep hasta que **bsp_sync** se ejecute; esto es incluso el caso para las variantes de comunicación sin buffer(es decir, **bsp_hpput**).

DRMA

Nombre:
Acceso de memoria remota síncrona de volumen
Descripción:
Una forma de realizar la comunicación de datos en el modelo BSP es utilizar los recursos de comunicación de acceso de memoria remota directa (**DRMA, Direct Remote Memory Access**). Algunas bibliotecas de programación paralela requieren que las estructuras de datos usadas en operaciones de DRMA tienen que ser efectuadas en las posiciones de memoria estáticamente asignadas. BSPLib no tiene esta restricción, que permite la comunicación en un ambiente heterogéneo seguro, y asigna la comunicación en cualquier tipo de estructura de datos contigua incluyendo pilas de datos asignados. Esto solo se alcanza permitiendo a un proceso manipular ciertas áreas registradas de una memoria remota que se ha hecho previamente disponible por el proceso correspondiente. En este procedimiento de registración, los procesos usan la operación **bsp_push_reg** para anunciar la dirección del comienzo de un área local que está disponible para el uso global remoto.

Nombre:
bsp_push_reg : Registra una estructura de datos como disponible para el acceso a memoria remota directa.
En C:

```
#include "bsp.h"
void bsp_push_reg(const void *iden, int size);
```

Descripción:
Un programa BSPLib consiste en *p* procesos, cada uno con su propia memoria local. La estructura SPMD de tal programa produce *p* instancias locales de las varias estructuras de datos usadas en el programa. Aunque estas *p* instancias comparten el mismo nombre, en general, no tendrán la misma dirección física. Debido a la distribución de la pila , o debido a la implementación sobre una arquitectura heterogénea, uno puede encontrar que las *p* instancias de la variable *x* se han distribuido en hasta *p* direcciones diferentes.

Para permitir que los programas BSPLib se ejecuten correctamente se requiere un mecanismo para relacionar estas direcciones creando asociaciones llamadas registros. Un registro se crea cuando cada proceso llama a **bsp_push_reg** y, respectivamente, proporciona la dirección y extensión de un área local de memoria. Ambos tipos de información son relevantes como proceso que puede crear nuevos registros por proporcionar la misma dirección, pero diferentes extensiones. La semántica adoptada por el registro permite procedimientos llamados dentro de los supersteps para ser escritos de una manera modular permitiendo registros más nuevos para remplazar temporalmente unos más viejos. Sin embargo, el esquema adoptado no impone el empaquetamiento estricto de los pares *push-pop* que se asocia normalmente a una pila. Esto proporciona las ventajas de la encapsulación proporcionadas por una pila, mientras proporciona la flexibilidad asociada con una disciplina basada en pilas. En línea con la semántica del superstep, los registros toman efecto hasta la próxima sincronización de barrera.

Nombre:
bsp_pop_reg : Remueve la visibilidad de una estructura de datos previamente registrada.
En C:

```
#include "bsp.h"
void bsp_pop_reg(const void *ident);
```

Descripción:
Cuando cada proceso llama a bsp_pop_reg y proporciona la dirección de su área local que participa en el registro, se destruye una asociación de éste. Se levantará un error de ejecución si estas direcciones (es decir, una dirección por proceso) no refieren a la misma asociación del registro. En línea con la semántica del superstep, el desregistro toma efecto en la siguiente sincronización de barrera.
Nombre:

bsp_put, bsp_hpput : Deposita un dato en una memoria de procesos remotos.
En C:

```
#include "bsp.h"
void bsp_put(int pid, const void *src, void *dst, int offset,
             int nbytes);
void bsp_hpput(int pid, const void *src, void *dst, int offset,
               int nbytes);
```

Descripción:
El propósito de **bsp_put** y **bsp_hpput** es proporcionar una operación semejante a memcpy disponible en la biblioteca de Unix *string.h*. Ambas operaciones copian un número específico de bytes, desde una dirección en una estructura de datos en la memoria local de un proceso, hacia las locaciones de memoria contigua en la memoria local de otro proceso. El factor que distingue entre estas operaciones es proporcionado por la opción del almacenamiento secundario.

Nombre:
bsp_get, bsp_hpget : Copia datos desde una memoria de un proceso remoto.
En C:

```
#include "bsp.h"
void bsp_get(int pid, const void *src, int offset, void *dst,
             int nbytes);
void bsp_hpget(int pid, const void *src, int offset, void *dst,
               int nbytes);
```

Descripción:
Las operaciones **bsp_get** y **bsp_hpget** entregan en la memoria local de otro proceso y copian datos remotos anteriormente registrados guardados ahí en una estructura de datos en la memoria local del proceso que lo inició.

BSPM

Nombre:
Paso de mensajes síncrono de volumen .
Descripción:
El acceso a memoria remota directo (DRMA) es un estilo de programación conveniente para cómputos BSP que se pueden analizar estáticamente de un modo directo. Esto es menos conveniente para cómputos donde los volúmenes de datos que se comunican en los superstep son irregulares y dependientes de los datos, y donde el que cómputo se realiza en un superstep depende de la cantidad y forma de los datos recibidos al comienzo de ese superstep. Un estilo de programación más apropiado en tales casos es el **paso de mensajes síncrono de volumen (BSMP, *Bulk Synchronous message passing*)**.

En BSMP, se provee una operación send sin-bloqueo para que entregue los mensajes a un buffer de sistema asociado con el proceso de destino. Se garantiza que el mensaje esté en el buffer de destino en el comienzo del siguiente superstep, y puede ser accesado por el proceso de destino sólo durante ese superstep. Si el mensaje no es accesado durante ese superstep se remueve del buffer. En el mantenimiento con la semántica del superstep BSP, los mensajes enviados al proceso durante el superstep no implica ordenamiento en el fin del recibimiento; un buffer de destino puede por lo tanto visualizarse como una cola los mensajes entrantes ingresan en la cola en un orden arbitrario y son accesados en el mismo orden.

En BSPLib, el paso de mensajes síncrono de volumen se basa en la idea de mensajes en dos partes, una parte de longitud fija llevando información de las etiquetas que puede ayudar al receptor a interpretar el mensaje, y una parte de longitud variable conteniendo la carga principal de datos. Se puede denominar a la porción de largo fijo como la etiqueta y a la porción de largo variable como la carga. La longitud de la etiqueta requiere ser fijada durante un superstep particular, pero puede variar entre supersteps.
Nombre:
bsp_set_tagsize : Tamaño de la etiqueta de un paquete BSMP.
En C:

```
#include "bsp.h"
void bsp_set_tagsize(int *tag_bytes);
```

Descripción:
Permite que el usuario fije el tamaño de la etiqueta, permitiendo el uso de etiquetas apropiadas para los requisitos de comunicación de cada superstep. Todos los procesos deben llamar colectivamente al procedimiento.

Nombre:
bsp_send : Transmite un paquete BSMP a un proceso remoto.
En C:

```
#include "bsp.h"
void bsp_send(int pid, const void *tag, const void *payload,
              int payload_bytes);
```

Descripción:
La operación **bsp_send** se utiliza para enviar un mensaje que consiste en una etiqueta y una carga útil a un proceso de destino específico. El proceso de destino podrá tener acceso al mensaje durante el superstep subsecuente. La operación **bsp_send** copia la etiqueta y la carga útil del mensaje antes de volver. Las variables de la etiqueta y de la carga útil pueden por lo tanto ser cambiadas por el usuario inmediatamente después de **bsp_send**.
Nombre:
bsp_qsize : Ve cuántos paquetes BSMP llegaron.
En C:

```
#include "bsp.h"
void bsp_qsize(int *packets, int *accum_nbytes);
```

Descripción:
La función **bsp_qsize** es una función de pregunta que retorna el número de mensajes que fueron enviados a ese proceso en el superstep previo y que aún no han sido ocupados por un **bsp_move**. Antes de que cualquier mensaje se ocupe por **bsp_move**, el número total de mensajes recibidos igual será enviado por alguna operación **bsp_send** en el superstep previo. La función también devuelve el tamaño acumulado de todas las cargas útiles del mensaje no ocupado. Se entiende ésta operación para ayudar al usuario a asignar una estructura de datos de tamaño apropiado para tener todos los mensajes que fueron enviados al proceso durante un superstep.
Nombre:
bsp_get_tag : Chequea la etiqueta en un paquete BSMP.
En C:

```
#include "bsp.h"
void bsp_get_tag(int *status, void *tag);
```

Descripción:
Para recibir un mensaje, se deben utilizar los procedimientos bsp_get_tag y bsp_move. El efecto de la operación bsp_get_tag es que el estado del argumento tiene asignado el valor -1 si el buffer del sistema está vacío. En caso contrario se convierte en la longitud de la carga útil del primer mensaje en el buffer. Esta longitud se puede utilizar para asignar una estructura de datos de tamaño apropiado para copiar la carga útil usando **bsp_move**.
Nombre:
bsp_move : Mueve un paquete BSMP desde la cola del sistema.
En C:

```
#include "bsp.h"
void bsp_move(void *payload, int reception_bytes);
```

Descripción:
La operación **bsp_move** copia la carga del primer mensaje en el buffer dentro de la carga, y quita ese mensaje del buffer. La variable reception_bytes especifica el tamaño del área de recepción cuando la carga se copió dentro. La mayor parte de los reception_nbytes serán copiados dentro de la carga.

Nombre:
bsp_hpmove : Un método simple para mover un paquete BSMP desde la cola del sistema.
En C:

```
#include "bsp.h"
void bsp_hpmove(void **tag_ptr_buf, void **payload_ptr_buf);
```

Descripción:
La operación **bsp_hpmove** es un método sin copia de recepción de mensajes que está disponible en mensajes con punteros tal como C. La función **bsp_hpmove** retorna -1 si el buffer de sistema está vacío. En caso contrario retorna el largo de la carga del primer mensaje en el buffer y:

a) coloca un puntero a la etiqueta en tag_ptr_buf;

b) coloca un puntero a la carga en payload_ptr_buf;

c) quita el mensaje del buffer entrante del sistema.

Comunicaciones colectivas

Algunos sistemas de paso de mensajes, tales como MPI, proporcionan primitivas para varios modelos especializados de comunicación que se presentan con frecuencia en los programas de paso de mensajes. éstos incluyen la difusión, dispersión, recolección, intercambio total, reducción, sumas de prefijo (exploración), etc. Estos modelos estándares de comunicación también se presentan con frecuencia en el diseño de los algoritmos de BSP. Es importante que tales modelos estructurados se pueden expresar convenientemente e implementar eficientemente en un sistema de programación BSP, además de las operaciones más primitivas tales como put y get que generan modelos de comunicación arbitrarios y no estructurados. Se puede apreciar un resumen en la tabla 1 .

Table 1: Comunicaciones colectivas: biblioteca de nivel 1

OPERACI'ON	*SIGNIFICADO*
bsp_bcast	Difusión desde un proceso a todos
bsp_fold	Reduce los datos con un operador asociativo
bsp_ s	Retorna *s*, el índice de Mflop/s del procesador
bsp_l	Retorna *l*, el índice de sincronización de barrera en flops
bsp_g	Retorna *g*, el rendimiento de la comunicación en flops/word

Nombre:
bsp_bcast : Transmite datos desde un proceso.
En C:

```
#include "bsp_level1.h"
void bsp_bcast (int bcast_pid, void *src, void *dst, int nbytes);
```

Descripción:
Una transmisión colectiva en la cual los datos src de tamaño nbytes en el procesador bcast_pid se copian en la estructura de datos registrada dst en todos los procesadores.

Si el programa se conecta a bspfront usando la opción -stat, entonces la fórmula de costo BSP para la transmisión se copia en el archivo **STAT.bsp**.

Nombre:
bsp_fold : Enlaza un operador asociativo entre un ítem de dato único tomado en cada proceso.
En C:

```
#include "bsp_level1.h"
void bsp_fold (void (*op) (void*, void*, void*, int*),
               void *src, void *dst, int nbytes);
```

Descripción:
Una operación de reducción colectiva que combina las copias locales del dato *src* en cada procesador, usando un operador binario *op*, asociativo. El resultado de la reducción se copia en *dst* en todos los procesadores.

Nombre:
bsp_l, bsp_s, bsp_g, bsp_nhalf : Averigua los parámetros de BSP.
En C:

```
#include "bsp_level1.h"
double bsp_s ();
double bsp_l ();
double bsp_g ();
int bsp_nhalf();
```

Descripción:
Estas funciones devuelven los parámetros de BSP para una máquina:

bsp_s
: Es la velocidad del cómputo de un proceso en flop/s (es el número de operaciones de coma flotante por segundo).

bsp_l
: Es el costo de la latencia de la sincronización en unidades de bsp_s. Es decir, si $s = 1000$, entonces cada procesador de la máquina tendría que hacer 1000 operaciones de coma flotante en el mismo tiempo que todos los procesadores toman en sincronizar la barrera.

bsp_g
: Es el número asintótico de flops/word requerido por todos los procesadores para comunicar simultáneamente un mensaje.

bsp_nhalf
: Es el número de las palabras requeridas para comunicarse tales que el rendimiento de la comunicación deteriora a 2g.

En la implementación actual de la biblioteca, los valores para cada uno de estos parámetros se leen en un archivo de base de datos, y no son calculados por la experimentación en el tiempo de ejecución.

Comandos de usuario

En la tabla 2 se muestra un resúmen de los comandos de usuario.

Table 2: Comandos de usuario

CLASE	*COMANDO*	*SIGNIFICADO*
Compilación	bspcc	Driver de compilación para los
		programas en C
	bspf77	Driver de compilación para los
		programas en Fortran 77
	bspc++	Driver de compilación para los
		programas en C++
Soporte de TCP/IP	bsplibd	Demonio del TCP/IP
Perfilación	bspcgprof	Herramienta de visualización
		de componentes Call graph
	bspprof	Herramienta de predicción del
		rendimiento
	bspsig	Estilo de composición prof(1)
Comandos miscelaneos	bsprun	Ejecuta un programa de BSPLib
	bsparch	Controla la configuración de BSP
		y el dispositivo de comunicaciones
	ipcclean	Limpieza general los recursos de
		comunicación entre procesos
	bspparam	Enumera los parámetros de la
		máquina BSP
Programación de texto	litToPgm	Convierte un archivo fuente
		escrito en el texto del programa
	litToTex	Convierte un archivo fuente escrito
		en LaTeX

Nombre:
bspfront: Driver de compilación del Toolset BSP para BSPLib.
Sinopsis:

```
bspfront ( opciones ) ( nombres de archivos )...
bspf77 ( opciones ) ( nombres de archivos )...
bspcc ( opciones ) (nombres de archivos )...
bspc++ ( opciones ) ( nombres de archivos )...
```

Descripción
Los programas que han sido escritos para la biblioteca de programación paralela BSPLib deben ser compilados con bspfront. Este driver de compilación lleva a cada archivo de entrada a través de alguna de las posibles fases de compilación: extracción de código desde un programa literado, compilación, ensamblado y enlace.

Por cada archivo de entrada, la fase con la que se comenzará se determina por su extensión:

.c
fuente C; preprocesar, compilar, ensamblar
.f
fuente Fortran 77; compilar, ensamblar
.cc
fuente C++; preprocesar, compilar, ensamblar
.c++
fuente C++; preprocesar, compilar, ensamblar
.lc
fuente literada C; desliterar, preprocesar, compilar, ensamblar
.lf
fuerte Fortran literada 77; desliterar, compilar, ensamblar
.lcc
fuente literada C++; desliterar, preprocesar, compilar, ensamblar
.lc++
fuente literada C++; desliterar, preprocesar, compilar, ensamblar

El comando **bspcc** es sólo un alias para **bspfront**, puesto que **bspc++** y **bspf77** tienen el mismo comportamiento operacional que **bspfront -f77** y **bspfront -c++** respectivamente.
Opciones.
Se deben separar las opciones: "-dr" es bastante diferente de "-d - r" . Cualquier opción que no haya sido reconocida por bspfront se deriva, directamente, a través del compilador implícito. A continuación se muestra un resumen de todas las opciones, agrupadas por tipo.

Opciones generales.

-v
Imprime (en la salida de error standard) los comandos ejecutados para hacer funcionar las etapas de compilación. También muestra el número de la versión del driver de compilación y las opciones de compilación estandar que están establecidas para su plataforma.
-c
Suprime la fase de carga de la compilación y fuerza a que se produzca un archivo objeto incluso si se compila solamente un programa.
-o Archivo
Coloca la salida en el archivo mensionado. Esto aplica indiferencia a cualquier orden de salida de bspfront que se esté produciendo, ya sea éste un archivo ejecutable u objeto.
-cflags opciones
Pasa las opciones directamente a través del compilador implícito C. Esta opción es útil cuando se requiere una opción en el compilador implícito C, pero ésta choca con una de las opciones de bspfront. Por ejemplo, -v se vuelve contra un modo difuso bspfront. Si se requiere el modo difuso del compilador implícito C (el cual también usa el flag -v) entonces use **-cflags '-v'.**
-f77flags opciones
Pasa las opciones directamente a través del compilador implícito Fortran 77.
-c++flags opciones
Pasa las opciones directamente a través del compilador implícito C++.
-overridec++ comando
Usa el comando como un compilador alternativo C++.
-overrideld comando
Usa el comando como un linker alternativo.

-keep

Cualquier archivo temporal, creado por el driver de compilación en TMPDIR, no se remueve hasta final de la compilación.

-showlog

Muestra una lista de los usuarios que han utilizado el driver de compilación. Esta opción sólo es válida si bspfront ha sido instalado para registrar su uso.

Opciones de preprocesamiento.

-E

Preprocesa sólo los archivos nombrados C y envía el resultado a la salida estándar. La salida contendrá directivas preprocesadas para el uso en el próximo paso del sistema de compilación.

-I*dir*

Agrega el directorio *dir* a la lista de directorios buscados para los archivos incluidos.

-D*macro*

Define macro *macro* con el string "1" como su definición.

-D *macro=defn*

Define la macro *macro* como *defn.*

-U*macro*

Causa que cualquier definición de macro sea indefinida, como si fuera una directiva de preprocesamiento #undef. Si se especifica el mismo nombre para **-D** y **-U**, no se define macro, a pesar del orden de las opciones.

Opciones de Perfil.

-stat

Presenta un informe sobre las estadísticas del desarrollo de un programa BSP y los costos teóricos de las comunicaciones colectivas. El archivo **STAT.bsp** será escrito en el directorio de trabajo actual.

-prof

Genera una información de perfilación de BSP que sirve para el programa de análisis bspprof. El archivo **PROF.bsp** será escrito en el directorio de trabajo actual.

-cgprof

Genera una información de perfilación de Call graph de BSP que sirve para el programa de análisis bspcgprof. El archivo **PROF.bsp** será escrito en el directorio de trabajo actual. Esta opción deberá ser usada durante la compilación y el enlace.

-f(no)trottle-procs

Cuando está encendido este flug, el número de los procesos que pueden ejecutar concurrentemente la parte local de cómputo de un superstep se limita a 1, o el número especificado por la variable de entorno BSP_THROTTLE_PROCS.

Opciones de optimización general.

-O -O2 o -O3

Activa los paquetes de optimisación apropiados en los compiladores implícitos C, C++ o Fortran 77.

-flibrary-level level

Solicita el nivel de integridad controlado para que sea desarrollado por la biblioteca. El nivel por omisión es 0.

- ◊ 0 realiza pruebas en el registro, fuera del límite de DRMA, y consistente en argumentos de comunicaciones colectivas. La comunicación primitiva, en este nivel de optimisación, puede que no sea óptima, pero se garantiza que es un recurso favorable.
- ◊ 1 es una versión de recurso favorable de la biblioteca. Esto significa que si más procesos (incluyendo a otros procesos de usuarios) están siendo ejecutados en la máquina con los procesadores, entonces no se deteriora la ejecución de la biblioteca.
- ◊ 2 es una versión mejorada de la biblioteca que ha sido adaptada de tal forma que sólo asume que el proceso más grande se está ejecutando en la máquina paralela.

-f(no)combine-puts

Donde es posible, combina los puts múltiples al mismo procesador en un solo put.

-fcombine-puts-buffer number(,max,min)

Cada proceso tiene buffers de un tamaño total de 2**number***(*P*-1) para puts combinados; donde *P* es el número de procesos. Si se especifica la opción *max*, entonces el tamaño total de combinaciones de buffers a través de todos los procesadores será *max*; y min el tamaño del buffer donde la optimisación que se apaga como *max*/*P* siendo más pequeña que *min*.

-fcontention-resolve level

Aplica optimisaciones al eliminar automáticamente los conflictos de los mensajes dentro de un algoritmo. Hay 3 niveles de optimisación diferentes (el nivel por omisión es 1)

- ◊ 0 no posee eliminación de conflictos.
- ◊ 1 siempre aplica la eliminación de conflictos.
- ◊ 2 sólo aplica la eliminación de conflictos a las transferencias almacenadas,es decir, no usa eliminación de conflictos en los puts de alto rendimiento (bsp_hpput), de manera que el put pueda ocurrir desincronizado del cómputo.

-ffixed-bsmp-buffers number

Utiliza los buffers del sistema de tamaño fijo para las operaciones del paso de mensajes síncrono de volumen. Algunos sistemas (particularmente el Cray T3D), tendrían buffers de tamaño arbitrario denotados por la opción -finfinite-bsmp-buffers incurriendo en una gran sanción en la latencia de la sincronización de barrera.

-finfinite-bsmp-buffers

Utiliza buffers BSMP, los cuales cambian dinámicamente de tamaño como se describe en el propósito BSPLib.

-bspbuffer number

Fija el tamaño del paquete para los mensajes de salida. Si el número termina en una "K", entonces el tamaño está en kilobytes; de otro modo se asume que el tamaño está en bytes. En algunos sistemas, este valor se fija por la implementación.

-bspfifo number

En esta implementación de BSPLib, todos los puts y los gets se posponen hasta el final del superstep. Esta opción fija el tamaño inicial de una cola interna FIFO, que mantiene todas las comunicaciones atrasadas. Si se emiten más puts durante un superstep, el FIFO se extiende en incrementos de tamaño numerado. En algunos sistemas, se utiliza un FIFO separado para mantener todos los gets, y esto no se puede extender durante el tiempo de ejecución. Esta fila sólo necesitará incrementarse si se provoca un desbordamiento en el tiempo de ejecución.

Opciones de optimización de memoria compartida.

-bspnobuffers number.

Cada proceso tiene un número de buffers entrantes, asociado con él. La comunicación de BSPLib ocurre en un estilo de paso de mensaje a través de estos buffers de memoria compartida. El resto de los buffers que existen, tienen mayor oportunidad de que el proceso de envío pueda enviar un mensaje al proceso de destino. Sin embargo, si se distribuyen demasiados buffers, entonces el proceso de recepción puede enviar servicios de buffers demasiado grandes que estarán vacíos. Esta opción fija el número de buffers de entrada asociado a cada proceso.

-bspbuffestalls number.

El número de buffers de entrada especificado por la opción -bspnobuffers se sirve de un estilo circular encadenado. Después se detiene en el número (es decir, nada entra), se prueba el dato para que sea enviado de una manera similar, es decir, las pruebas de los números se hacen para enviar paquetes.

Opciones de optimización de redes de workstation.

-bspslotsize *varepsilon*.

En BSPLib, el mecanismo de paso de la capa de transporte se alcanza usando una forma de multiplicidad estadística de división de tiempo, que trabaja de la siguiente manera. El tamaño de la estructura ((**-bspbuffer)** y el número de procesadores **(bsp_nprocs())** involucrados en la comunicación son conocidos. Como los cronómetros de los procesadores no están necesariamente sincronizados, no es posible permitir el acceso a los procesadores de acuerdo a alguna permutación, siendo una técnica aplicada satisfactoriamente en más arquitecturas estrechamente unidas. De esta forma los procesadores eligen un canal, *q*, uniformemente al azar en el intervalo (0... Q-1) (donde *Q* es el número de procesadores comunicándose al final de un superstep particular), y proyecta su transmisión para este canal. La elección de un canal aleatorio es importante si los cronómetros no están sincronizados, ya que esto asegura que los procesadores no optarán por una mala comunicación repetitivamente. Cada procesador espera el tiempo *q* * *varepsilon* microsegundos después del comienzo del ciclo, donde *varepsilon* es un canal de tiempo, antes de pasar a otro paquete a la capa de transporte. La longitud del canal, *varepsilon*, se elige basandose en el

tiempo máximo que el canal puede ocupar el medio físico, y toma en consideración las colisiones que pueden ocurrir cuando se alcanza un buen rendimiento . El mecanismo se diseña para permitir al medio a operar en el estado seguro que alcanza un alto rendimiento. Desde que el arranque de comunicación ha sido igualado por este protocolo canalizado, se prohibe el comportamiento irregular del protocolo de bajo-nivel, y se asegura una amplia utilización del medio.

-bspnoslots *number*.
Esta opción especifica el número de canales de tiempo requerido para los procesadores **bsp_nprocs()** para enviar un mensaje simultáneamente a otro proceso. Para un bus compartido o repetidor *number* debería ser igual a **bsp_nprocs().** Para un interruptor con enlaces dobles completos y una capacidad posterior que es la suma de las capacidades de cada uno de los enlaces, se debe fijar *number* a uno.

-bsproundtrip number.
El número es el tiempo de demora de un viaje de ida y vuelta, en microsegundos, de un mensaje que no contiene carga útil.

-bspsendlatency number.
En el estado seguro, donde el proceso emite una corriente de paquetes vacíos dentro de la red, el número es el tiempo, en microsegundos, en dichos paquetes se pueden emitir. Como usualmente éste es más rápido para emitir paquetes que para recibirlos, el tiempo probablemente igualará la razón de envío de un paquete, como un estado seguro se puede lograr sólo cuando los procesos están enviando y recibiendo paquetes a la misma razón. El parámetro debería ser usado para reducir la velocidad de un emisor rápido, para que emita paquetes en un grado en el cual el receptor pueda, con seguridad, enviar estos paquetes.

Opciones de enlace.
Estas opciones funcionan cuando el compilador enlaza los archivos objeto en un archivo de salida ejecutable. Ello no tienen sentido si el compilador no hace un paso de enlace.

object-file-name.
Se considera un nombre de archivo que no termina en un sufijo especial reconocido para nombrar un archivo objeto o librería. (Los archivos objeto se distinguen de las bibliotecas por el linker, de acuerdo a los contenidos del archivo). Si GCC hace un paso de enlace, estos archivos son usados como entrada para el linker.

-device device.
Determina qué dispositivo de comunicación debería usarse para los programas de BSPLib. El dispositivo por omisión se especificada por el comando **bsparch -device.** La opción de dispositivo tiene que ser un dispositivo válido de BSPLib, y tiene que instalarse ese dispositivo.

-mpi.
Se escribe para la opción *-device MPASS_MPI.*

-udpip.
Se escribe para la opción *-device MPASS_UDPIP.*

-tcpip.
Se escribe para la opción *-device MPASS_TCPIP.*

-shmemsysv.
Se escribe para la opción *-device SHMEM_SYSV.*

-l*library*.
Usa la biblioteca nombrada cuando enlaza.

El linker busca una lista estándar de directorios para la biblioteca, la cual es, realmente, un archivo de nombre **lib*library*.a.** El linker utiliza este archivo como si hubiese sido especificado previamente por el nombre.

Los directorios buscados incluyen varios directorios de sistema estándar más cualquiera que se especifique con **-L.**

Normalmente los archivos encontrados de esta forma son archivos de la biblioteca, cuyos miembros son archivos objeto. El linker toca un archivo para examinar a través de éste a los miembros, quienes definen unos símbolos que han sido referenciados pero no definidos. Sin embargo, si el linker encuentra un archivo objeto ordinario más que una biblioteca, el archivo es enlazado en el modo usual. La única diferencia entre usar una opción **-l** y especificar un archivo de nombre es que **-l** rodea a la biblioteca con **lib** y **.a** y busca varios directorios.

-c++
El compilador de C es el linker por omisión. Esta opción cambia de linker al compilador de C++.

-f77

Usa el compilador de Fortran 77 como linker. La opción **-ldflags flags** pasa los flags directamente a través del linker.

Opciones de directorio.

Estas opciones especifican directorios para buscar archivos de encabezado, bibliotecas y partes del compilador.

-Idir

Agrega el directorio *dir* a la lista de directorios buscada por los archivos *include.*

-Ldir

Agrega el directorio *dir* a la lista de directorios buscada por las bibliotecas.

Nombre:
bsplibd Demonio para los programas de BSPLib ejecutados sobre TCP/IP y UDP/IP.
Sinopsis:

```
bsplibd [-all]
```

Descripción:
El propósito de éste comando es comenzar dos demonios, bspnowd y bspportd, que permite a una máquina involucrarse en un cómputo BSPLib cuando usa las versiones TCP/IP o UDP/IP de la biblioteca. Estos demonios controlan la partida de los programas en nodos remotos:

bspnowd (demonio del proceso de usuario)

. El demonio bspnowd controla el acceso a una máquina para un usuario en particular (este demonio se usa para comenzar procesos en el nodo local como un resultado de un requerimiento de partida remota). La seguridad se controla permitiendo que los procesos solo sean iniciados si el user-id y la password en la máquina que inicia el trabajo BSP es la misma que en la máquina en la cual el demonio usuario se está ejecutando. El demonio es comenzado por un usuario que ejecuta el comando bsplibd en cada máquina en un grupo. Si un demonio para el usuario ya se está ejecutando en una máquina, entonces comienza un nuevo demonio que podría apagar el primero (y matar todos los trabajos BSP no terminados para ese usuario) antes que comience el nuevo.

bspportd: demonio del puerto

. El demonio bspportd mantiene una correspondencia entre los números de puerto y los demonios de usuarios de BSPLib. Hay solo un demonio de puerto por máquina (puede ser comenzado por cualquiera), aunque haya un bspnowd por usuario. Este demonio también se comienza ejecutando el comando bsplibd en cada máquina en un grupo. Si el demonio de puerto ya está activo (es decir, comenzado por otro usuario), entonces no se necesita comenzar otro demonio, y un error (que se puede ignorar) puede reportar las fallas enlazadas:

```
pine.comlab> bsplibd
BSP/NOW Port Daemon(pine): bind() call failed:
Address already in use
BSP/NOW Daemon (pine): forked with pid 26840.
```

Una forma de comenzar los demonios en todas las máquinas especificadas en un grupo, y establecer un entorno correcto de BSPLib para un único usuario es la siguiente:

```
pine.comlab> bsplibd -all
pine.comlab> bspload -start -all
```

Esta forma requiere que pine tenga una entrada en los archivos .rhost en todas las máquinas en un grupo, como esta usa rsh. Si rsh no est disponible en la plataforma, se tendrá que comenzar los demonios en cada una de las máquinas individualmente. Los demonios solo necesitan comenzar una sola vez, y esto establecer el entorno para todos los trabajos subsecuentes de BSPLib para ese usuario. El usuario no necesita registrarse sobre cada una de las máquinas una vez que han comenzado los demonios. Alguna interacción del usuario con los demonios se puede ejecutar remotamente desde una única workstation de un grupo BSPLib.

El manejador de carga (bspload), el programa que comienza un proceso de BSP (bsprun), y el sistema de paso de mensajes TCP/IP o UDP/IP usado en la implementación de BSPLib requieren que un usuario específico del grupo de workstation sea definido en el archivo /.bsptcphosts. Como un ejemplo de un archivo, consideremos un grupo de cuatro workstation llamadas pine, ash, oak y mercury, algunas de las cuales tienen multiprocesador SMP, y dos de ellas están conectadas por un anillo FDDI separadas del laboratorio normal de redes Ethernet:

```
pine.comlab> cat ~/.bsptcphosts
host(oak);
host(pine);
host(ash)  ncpus(2) adapter(ash-fddi);
host(mercury)  ncpus(4) adapter(mercury-fddi);
```

El archivo contiene un número de descriptores cada uno delimitado por un punto y coma. El primer descriptor en una línea tiene que ser el host, y tiene que ser un nombre de servidor preguntable para la máquina. Las otras entradas son opcionales, pero cambia el comportamiento del grupo BSP. Por ejemplo, la clave ncpus especifica el número de procesadores en una workstation. Esta información es usada por bspload para que múltiples procesos puedan comenzarse en el mismo procesador.

Si un host no se ha indicado en la lista de los host, entonces no se considera parte del grupo. Desde que el demonio de carga mantiene las cargas para toda la comunidad de usuarios, la lista de host usada al ser ejecutado el demonio de carga podría ser un superconjunto de todas las listas de host de los usuarios.

Para compilar un programa BSP para una ejecución TCP/IP, éste se debe compilar con bspfront y la variable de entorno BSP_DEVICE debe fijarse en MPASS_TCPIP; y al compilar uno para UDP/IP, se debe compilar con bspfront y la variable de entorno debe fijarse en MPASS_UDPIP. El comando bsprun se usa para iniciar un trabajo de BSP. Por omisión, este comienza el número requerido de procesos en el procesador con menor carga en una red. Se pueden usar las siguientes opciones para anular este comportamiento:

bsprun -noload
: Ignora el demonio de carga, y los procesos se comienzan en los procesadores en el orden de la lista.

bsprun -local
: Garantiza que los procesos BSP con bsp_pid() igual a cero se ejecuten en el procesador en el que bsprun fue ejecutado. Esto puede ser requerido para propósitos de entrada/salida.

bsprun -nolocal
: Los procesos BSP no serán ejecutados en el procesador que comenzó con bsprun.

Un trabajo de BSP no necesita ejecutarse en un grupo de workstations que tienen un archivo de sistema NFS en común. Para grupos sin NFS, los sistemas TCP/IP y UDP/IP asumen que la estructura de directorio sobre **$HOME** es la misma en cada máquina del grupo.

Opciones:

-v
: Imprime(en la salida de error estándar) las fases involucradas en la ejecución del comando.

-help
: Imprime el manual en la salida estándar.

-all
: Si la máquina en la que se ejecuta el comando bsplibd tiene una entrada en los archivos .rhost de todas las máquinas del grupo BSP, entonces los demonios serán comenzados en todas las máquinas listadas en .bsptcphosts.

Nombre:
bspcgprof Perfilador Call-Graph de Oxford BSP Toolset para BSPLib.
Sinopsis:

```
bspcgprof [-v] [-help] [-hide percentage] filename
```

Descripción:
El comando bspcgprof visualiza el gráfico desde la ejecución de un programa BSPLib. Los archivos trazados por Call-Graph Se generan agregando la opción -cgprof cuando se compilan los programas BSPLib con bspfront.
Opciones:

-v
Imprime (en la salida de error estándar) las fases involucradas en la ejecución del perfilador. La opción también imprime la versión del perfilador.
-help
Imprime el manual en la salida estándar.
-hide percentage
Contrae todos los nodos en el gráfico que tiene un cómputo acumulado y un menor tiempo de comunicación que el porcentaje del tiempo total de ejecución.

Nombre:
bspprof: Perfilador de comunicación del Toolset de BSP para BSPLib.
Sinopsis:

```
bspprof [input-file] [output-file] [command-line-options]
```

Descripción:
El perfilador de comunicación de Oxford BSP Toolset analiza los archivos indicados generados durante la ejecución de un programa BSPLib. El perfilador convierte los archivos señalados en documentos PostScript que son apropiados para su visualización en una amplia gama de dispositivos gráficos. Los archivos señalados se generan agregando la opción -prof cuando se compilan los programas de BSPLib con bspfront.

El perfilador produce tres tipos de gráficos PostScript como salida:
1) Estrato-gráficos para el número de comunicaciones distintas que ocurren en el tiempo, o para el número de bytes comunicados. Dos gráficos se representan por perfil:
a) El número de bytes (o comunicaciones) entrando a un procesador.
b) el número de bytes (o comunicaciones) saliendo desde un procesador.
2) Los gráficos de predicción son similar a 1) excepto que el gráfico más bajo muestra los costos predichos de la comunicación en la máquina en la que el perfilador fue ejecutado. Si el se usó el comando -arch para especificar otra arquitectura, entonces esos parmetros de las arquitecturas BSP en cambio serán usados. Alternativamente, los parmetros BSP de la arquitectura que es para la predicción se pueden ingresar usando la opción -slg.
3) Un gráfico G, que también se basa sobre 1) excepto que el gráfico de abajo muestra el efecto de *g* sobre el tiempo. Para una máquina BSP verdadera, *g* debería ser constante.

Nombre:
bspsig. Genera una señal desde el perfil de BSP.
Sinopsis:

```
bspsig [-v] [-help] [-wfm] [-sig] [-stat] [-sort key]
       [-bucket size] [-compress size] [-ignore percentage]
       [-slg bsp-parameters] filename
```

Descripción:
Genera un perfil y una señal del costo para el nombre de archivo indicado. Los archivos indicados se generan agregando la opción -prof cuando se compilan los programas BSPLib con bspfront.

Nombre:
bsprun Inicia la ejecución paralela de un trabajo BSP.
Sinopsis:

```
bsprun[ -v ] [ -npes number ] [ -local ] [ -nolocal ]
      [ -noload ] [ -splitio ] program command-line-options
```

Descripción:
El propósito de este comando es ocultar las especificaciones de la máquina al comienzo del trabajo paralelo. El programador especifica con la opción -npes un límite superior del número de procesos paralelos que serán producidos por el programa. El valor no anular el valor especificado por bsp_begin.
Opciones:
-v
Imprime (en la salida de error estándar) las fases involucradas en la ejecución de un comando.
-help
Imprime su manual en la salida estándar.
-local
En las implementaciones de la biblioteca con TCP/IP o UDP/IP, esta operación fuerza a que el proceso cero esté en el procesador que inicie el cómputo.
-nolocal
En las implementaciones de la biblioteca con TCP/IP o UDP/IP, esta opción fuerza al proceso cero esté en algún otro procesador que no sea el que inicie el cómputo.
-noload
En las implementaciones de la biblioteca con TCP/IP o UDP/IP, el demonio de carga no ser usado para determinar cual procesador estar produciendo. Los procesadores sern probados en el orden especificado en el archivo /.bsptcphosts.
-npes
number Produce el mayor número de procesos BSP en paralelo. Si hay *P* procesos físicos paralelos disponibles, y *p* procesos son generados por bsp_begin, entonces los *p* procesos son distribuidos a través de *P* procesadores equitativamente (cuando 1<=*p*<=*number*).
-splitio
Esta opción se propone ayudar a que el usuario determine cuál proceso imprimió datos en la salida estándar o error. Un mensaje de la forma:

```
--- pine.comlab.ox.ac.uk BSP pid 2 - stdout ---
```

se imprimirá cuando el pid de la fuente o flujo cambie. Esta opción solo está disponible para algunas plataformas.

Nombre:
bsparch Determina la arquitectura de la máquina y el dispositivo de comunicación.
Sinopsis:

```
bsparch [-arch] [-device] [-nodefault] [-name] [-defdevice machine]
        [-convertname machine] [-convertdevice device] [-allarch]
        [-alldevice] [-help]
```

Descripción:
El comando bsparch determina las opciones de instalación específicas de la máquina requeridas al instalar el Toolset de Oxford BSP. Dos factores guían la instalación del Toolset: (1) la arquitectura de la máquina y la versión del sistema operativo; y (2) el tipo de dispositivo usado como medio de comunicación. La opción -arch determina la arquitectura de la máquina, y -device el dispositivo de comunicación.

A la fecha, las arquitecturas probadas son:

SGICHALL64
- Power Challenge de Silicon Graphics (Irix 6.x)
SGIORIGIN
- Origin 2000 de Silicon Graphics (Irix 6.x)
SGI
- workstation de Silicon Graphics (Irix 5.x)
SP2
- SP2 de IBM ejecutando AIX
CRAYT3E
- T3E de Cray

CRAYT3D
- T3D de Cray

PARSYTEC
- Explorer de Parsytec

CONVEXSPP
- SPP de Convex

SunOS
- SunOS 4.1.x

Solaris
- SunOS 5.x

LINUX
- PC basado en Unix

OSF1
- alpha boxes de Digital

HITACHIMPP
- Hitachi SR2001

Los dispositivos de comunicación probados se muestran abajo. Se puede notar, que solo un pequeño subconjunto del producto cruz de arquitecturas y dispositivos son construcciones válidas para el Toolset

SHMEM_SYSV
- System V shared memory.

SHMEM_SGI
- SGI specific shared memory operations.

MPASS_TCPIP
- TCP/IP based message passing.

MPASS_MPI
- MPI.

MPASS_MPL
- IBM's message passing system.

MPASS_PARIX
- Message passing on the Parsytec GC.

MPASS_EXPRESS
- Express (Parasoft Ltd) message passing.

DRMA_SHMEM
- Cray Shmem one-sided communications.

El funcionamiento del dispositivo por omisión de bsparch puede anularse al proporcionar una variable de entorno BSP_DEVICE en el entorno del usuario, cuyo valor es el nombre del dispositivo señalado. Se podría anular el funcionamiento por omisión cuando se instala BSPLib al comienzo de MPI.

Nombre:
ipcclean Limpia los medios de comunicación entre procesos.
Sinopsis:

```
ipcclean [ -v]
```

Descripción:
El comando ipcclean remueve cualquier medio de comunicación entre-procesos izquierdo dejedo por un usuario del programa. La opción -v muestra la memoria compartida y los semáforos que fueron limpiados.

Nombre:
bspparam Imprime una tabla de los parámetros BSP de la máquina.
Sinopsis:

```
bspparam [ -v ] [ -help] [ ] [ -time]
```

Descripción:

Este comando genera una tabla conteniendo los parámetros de costo BSP para la máquina. Las entradas en la tabla se leen desde una base de datos y no generan demanda.

La base de datos de parámetros BSP es usada por bspprof, bspsig, y las funciones de bsplib_level1: bsp_s, bsp_l, bsp_g y bsp_nhalf. Estas funciones se usan en la definición de las operaciones de comunicación colectiva bsplib_level1 como una implementación alternativa de una operación colectiva que puede cambiar en el tiempo de ejecución dependiendo sobre los valores de algún parámetro BSP de la máquina.
Opciones:

-v
: Imprime (en la salida de error estándar) las fases involucradas en la ejecución del comando.

-help
: Imprime el manual en la salida estándar.

-time
: Los parámetros BSP se dan en microsegundos en vez del número equivalente de operaciones de punto flotante.

-a
: Genera entradas para todas las máquinas almacenadas en la base de datos de los parámetros BSP.

Nombre:
litToPgm Convierte un archivo literado en un programa.

litToTex Convierte un archivo literado en LaTeX.
Sinopsis:

```
litToPgm archivo-de-entrada
litToTex archivo-de-entrada
```

Descripción:
Un estilo de programación literado adopta un enfoque inverso a los comentarios. Cada cosa en un archivo literado se considera como un comentario, a menos que se especifique como código del programa. El comando litToPgm convierte un documento literado LaTeX en una forma adecuada para entrar en un compilador C o Fortran. el código se identifica por el entorno especial de LaTeX `\begin{code} \end{code}`. El comando litToTex convierte un documento literate LaTeX en un documento vanilla LaTeX. El archivo modelo lit.sty se puede incluir en su documento como contener la definición del entorno `\begin{code} \end{code}`.

About this document ...

This document was generated using the **LaTeX**2HTML translator Version 98.1p1 release (March 2nd, 1998)

The command line arguments were:
latex2html `-split 0 -no_navigation bsplib.tex`.

The translation was initiated by on 2000-08-22

2000-08-22