Factores de rendimiento en aplicaciones híbridas (MPI+OpenMP)

Abel Castellanos[1], Eduardo César[1], Anna Sikora[1], Joan Sorribes[1] , Andreu Moreno Vendrell[2] , Tomás Margalef[3]

Resumen— **En la actualidad, existe una amplia diversidad de modelos de programación para el desarrollo de aplicaciones paralelas. Cada uno de estos modelos fueron diseñados con el fin de aprovechar las características propias de la arquitectura para las que fueron diseñados. En particular, las aplicaciones híbridas (MPI-OpenMP) son aquellas que utilizan el paso de mensajes entre los diferentes procesos de la aplicación a través de librerías como openMPI o MPICH, y el paralelismo de memoria compartida dentro de cada nodo de cómputo defido en el programa por los pragmas OpenMP. Nuestro trabajo, se concentra en identificar los principales factores de rendimiento para estas aplicaciones de manera que, un ajuste de los mismos de manera estática o dinámicamente, podría provocar una mejoría sustancial en el rendimiento y la eficiencia de las mismas. Nos concentramos en el estudio de estos factores asumiendo que las aplicaciones son ejecutadas en un clúster homogéneo.**

En adición, se analiza detalladamente como se expresan estos factores en aplicaciones híbridas tipo Master-Worker y Pipeline.

Palabras clave— **MPI, OpenMP, Aplicaciones híbridas, Factores de rendimiento.**

I. Introducción

DEntro de la comunidad científica que investiga en torno a la Computación de Altas Prestaciones, una parte importante de los esfuerzos están dirigidos al modelado y evaluación del rendimiento de aplicaciones para las diferentes arquitecturas existentes en la actualidad. Con la aparición, en los últimos años de los procesadores con varios núcleos, se ha acuñado el término aplicaciones híbridas para aquellas aplicaciones que utilizan a la vez distintos modelos de programación paralelos para la implementación de dichas aplicaciones.

En la figura 1, se observa una clasificación general para la aplicaciones híbridas basada en la propuesta inicial [1] y como están relacionadas conceptualmente con los modelos tradicionales de programación paralela. Por un lado, tenemos las aplicaciones MPI [2] puras, donde el paralelismo esta expresado a partir del modelo de paso de mensajes entre los diferentes procesos involucrados. Por otro lado, tenemos las aplicaciones OpenMP [3] [4] puras, en donde el nivel de paralelismo se realiza dentro de cada nodo de cómputo. En este caso, no se realiza paso de mensaje entre los threads involucrados ya que los mismos comparten una memoria común. La creación, destrucción y sincronización de estos threads es realizada por la librería en el inicio y fin de las secciones paralelas o son especificadas explícitamente por el programador.

En particular, las aplicaciones híbridas MPI+OpenMP son una alternativa para aprovechar las bondades del paralelismo a ambos niveles. De esta forma, un proceso de una aplicación híbrida de manera general, estaría compuesto por una región serie, una región de comunicación entre procesos y una región paralela definida por los pragmas OpenMP.

Una descripción aun más detallada de este tipo de aplicaciones, debe considerar las ubicación de las comunicaciones dentro de la lógica de ejecución de nuestro programa y si estas se solapan con las fases de cómputo de la aplicación. De esta forma, tendríamos las aplicaciones híbridas modelo Masteronly, las Masteronly-single overlap y las aplicaciones con solapamiento múltiple cómputo - comunicación en los threads utilizados.

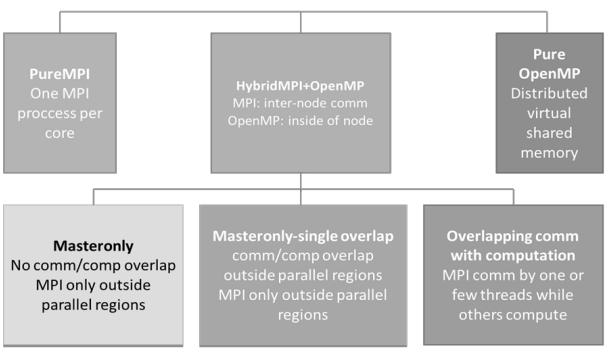

Fig. 1. Clasificación de las aplicaciones híbridas MPI+OpenMP

En el primero de los casos, las aplicaciones Masteronly, las llamadas a funciones de comunicación se realizan fuera de la región paralela por lo que el solapamiento entre regiones de cómputo y de comunicación solo se observará entre los diferentes procesos involucrados. A diferencia, las aplicaciones Masteronly-singleoverlap mantienen la comunicación fuera de las regiones paralelas pero al ser la mismas asíncronas, se obtendría un solapamiento cómputo-comunicación adicional dentro del proceso MPI. El último caso, las aplicaciones con múltiple solapamiento cómputo-comunicación, agrupa aquellas aplicaciones donde

[1]Dpto. de Arquitectura de Computadores y Sistemas operativos, Univ. Autónoma de Barcelona, e-mail: acastellanos@caos.uab.es, eduardo.cesar@uab.es, ania@caos.uab.es, joan.sorribes@uab.es

[2]Escuela Universitaria Salesiana de Sarria (EUSS), e-mail: amoreno@euss.cat

[3]Dpto. de Arquitectura de Computadores y Sistemas operativos, Univ. Autónoma de Barcelona, e-mail: tomas.margalef@uab.es

la comunicación es realizada dentro de la región de cómputo paralelo. En este caso tendríamos solapamiento a ambos niveles: entre procesos diferentes y dentro del cada proceso.

II. Factores de rendimiento

Basados en los modelos de rendimiento desarrollados previamente para aplicaciones MPI puras de tipo Master-Worker [5] y Pipeline [6], hemos identificado algunos factores adicionales que consideramos determinantes en el rendimiento de este tipo de aplicaciones, ahora con la inclusión de la regiones de computo paralelo OpenMP en cada proceso MPI. Para ello nos orientamos por una propuesta de metodología para el modelado de aplicaciones híbridas [7] cuya finalidad, es alcanzar un modelo general de rendimiento para las mismas.

En resumen, el conjunto de factores de rendimiento a considerar serían los siguientes:

Balance de la carga de cómputo: De manera similar a las aplicaciones MPI puras, un desbalance de la carga de trabajo entre los diferentes procesos de la aplicación provocaría una degradación del rendimiento en las aplicaciones. Gracias a la existencia de los dos niveles de paralelismo, las aplicaciones híbridas disminuyen, en cierta medida esta influencia pero no en la suficiente medida como para obviar la importancia de dicho factor.

Cantidad de procesos MPI: Cantidad de procesos MPI utilizados por la aplicación en su ejecución.

Cantidad de threads por cada proceso: Define la cantidad de threads utilizados en las regiones de cómputo paralelo OpenMP dentro de cada procesos de la aplicación. Este factor no se puede desligar de la cantidad de procesos MPI ya que, la decisión más adecuada sería la búsqueda de una combinación de ambos factores en la que nos acerquemos al rendimiento óptimo de la aplicación para una carga de trabajo determinada.

Afinidad: Describe cual es la política de asignación de los *cores* de nodo de cómputo a los threads utilizados en las regiones paralelas OpenMP. Podemos definir una política en la cual se asignen los *cores* mas cercanos entre si de manera que los mismos compartan la memoria cache o todo lo contrario: intentar que se comparta lo mínimo posible entre los *cores* la memoria cache a sus diferentes niveles. En todo caso, la decisión pasaría por evaluar/predecir el rendimiento de la aplicación para ambas políticas de afinidad, una cantidad de threads a utilizar y una carga de trabajo determinada.

Patrón de comunicación: En las aplicaciones SPMD es usual encontrar que las comunicaciones entre los procesos siguen una regularidad. Dicho comportamiento de manera gráfica (Figura 2), muestra como los procesos se comunican con los procesos vecinos para intercambiar información.A medida de que aumenta dimensión, la cantidad de mensajes enviados estre los procesos también aumenta aunque ello suele implicar que el tamaño de los mismos disminuya. Dependiendo de las características propias de la aplicación, este aumento podría beneficiar o perjudicar el rendimiento. En las aplicaciones Master-Worker y Pipeline, no tiene sentido analizar la influencia de este factor, ya que el mismo está determinado por la cantidad de procesos involucrados en la ejecución.

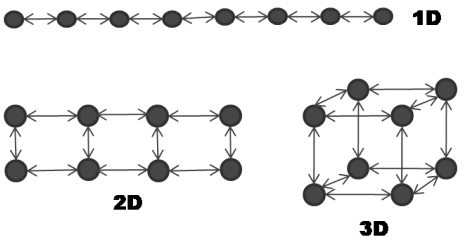

Fig. 2. Ejemplo de Patrones de comunicación en aplicaciones SPMD.

Calidad del paralelismo OpenMP: Es el único de los factores que no puede ser sintonizado en tiempo de ejecución. El mismo describe la influencia de una buen paralelismo en estas regiones aprovechando, en lo posible, las características propias del hardware en que es ejecutada la aplicación. Lamentablemente, una mejora en el rendimiento de estas regiones paralelas a partir de la modificación de su código, no es una tarea trivial ya que requiere un esfuerzo añadido del programador en búsqueda de modificaciones que efectivamente contribuyan a este objetivo.

Carga de trabajo de los threads de comunicación: Es un factor solo presente en las aplicaciones híbridas con múltiple solapamiento. Para un buen ajuste de este factor, se debería disminuir la carga de trabajo de los threads que realizan comunicación y cómputo con relación a los que realizan únicamente cómputo en la región paralela ya que inicialmente los primeros tendrían un tiempo de ejecución superior (Fig. 3). De esta forma, se conseguiría balancear el tiempo de ejecución entre todos los threads de la región paralela. Es investigaciones previas [8], se ha conseguido mejorar de manera notable en rendimiento de las aplicaciones paralelas gracias a la adaptación de la lógica las mismas para permitir este solapamiento múltiple.

Para la aplicaciones Master-Worker y Pipeline, del listado original de factores identificados podemos descartar el patrón de comunicación y la carga de trabajo de los threads de comunicación. En el primer caso, el motivo radica en que el patrón

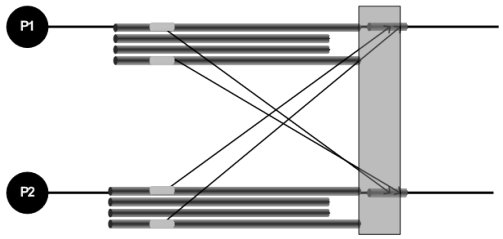

Fig. 3. Desbalance entre threads en las aplicaciones híbridas con múltiple solapamiento

de comunicación es rígido y está definido por el resto de los factores de rendimiento, por lo tanto no tiene sentido incluirlo como un factor adicional. En el caso de la carga de trabajo de los threads de comunicación, solo podríamos tomarlo en cuenta si la lógica de la aplicación permite realizar la comunicación entre los diferentes procesos MPI antes que se termine de realizar el cómputo paralelo para una fase de la misma. Este comportamiento no suele ser el característico en las aplicaciones Master-Worker y Pipeline ya que en ambos casos, la fase de envío de la información solo se puede realizar cuando se ha completado todo el cómputo asociado a esta fase.

A continuación (Fig. 4) mostramos tres configuraciones diferentes de ejecución para una aplicación Master-Worker. La figura nos muestra tres posibles decisiones que podríamos tomar con el objetivo de mejorar el rendimiento. En el primer caso, hemos configurado la aplicación para que sea ejecutada utilizando solo 3 workers en donde el proceso master utiliza un solo thread en su región de cómputo y los workers, a su vez, 3 threads. En el segundo caso, tenemos 5 workers con 2 threads por cada uno y 3 threads utilizados por el proceso Master. En el último caso, utilizamos 2 threads en el master, 4 workers y 4 threads por cada uno. Dependiendo de la aplicación, de las características de la arquitectura del nodo y de la red de interconexión, una configuración u otra podrían ser la mejor opción para obtener un rendimiento cercano al óptimo.

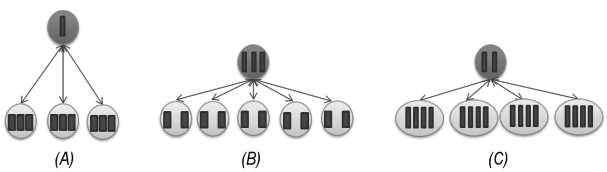

Fig. 4. Configuraciones de ejecución de una aplicación Master-Worker

Este mismo problema, lo podemos encontrar en las aplicaciones tipo Pipeline (Fig. 5). En este caso y a partir de la expresiones desarrolladas previas para aplicaciones Pipeline MPI, si una aplicación Pipeline tiene un etapa identificada como crítica (A) ya que es la de una duración significativamente superior el promedio del resto, la misma puede ser replicada en varios procesos similares donde tratando de distribuir la carga de trabajo entre ellos de manera homogénea. En este caso, las decisiones a la hora de configurar pueden ser varias: replicar el proceso en dos utilizando en cada uno 3 threads en la región de cómputo (B) o utilizar más procesos y diminuir la cantidad de threads como se muestra en e (C).

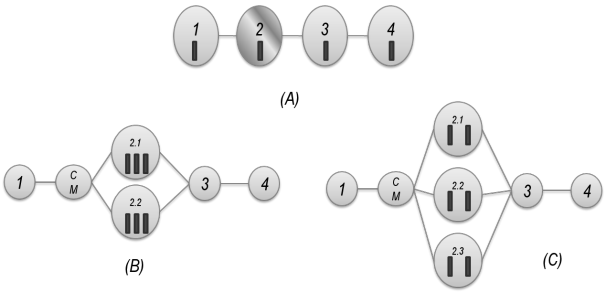

Fig. 5. Configuraciones de ejecución de una aplicación Pipeline (Etapa más lenta)

Otra posible decisión puede estar asociada a la forma en que se agrupan etapas rápidas (donde el tiempo de ejecución es inferior al promedio de los tiempos de todas las etapas). Para esta situación (Fig. 6), los procesos pueden ser agrupados en una misma etapa del Pipeline decidiendo a su vez, que cantidad de threads sería conveniente utilizar en cada uno con el fin de mejorar en rendimiento general de la aplicación. Cuantos procesos pueden ser agrupados y la cantidad de threads que utilizarán cada uno, serían las incognitas a solucionar para este tipo de aplicaciones en la búsqueda de uan configuración apropiada para alcanzar un mejor rendimiento.

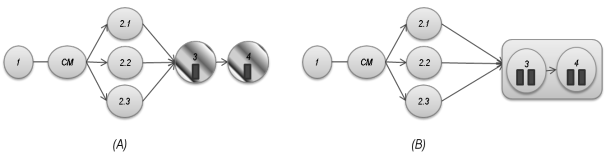

Fig. 6. Configuraciones de ejecución de una aplicación Pipeline (Etapas rápidas)

III. Resultados experimentales

Para validar la influencia de la cantidad de procesos e threads, así como de la afinidad fueron utilizadas dos aplicaciones. La primera de ellas, es una aplicación sintética de algebra matricial cuyo comportamiento sigue un patrón Master-Worker. El proceso Master se encarga de ir construyendo las matrices a partir de la expresión en formato post-fijo introducida como parámetro de ejecución de la aplicación y de enviar los fragmentos respectivos de cada matriz a cada uno de los procesos worker en donde se realiza el cómputo de los fragmentos respectivos de la solución final.

La segunda aplicación fue desarrollada por nuestro departamento con el fin de agrupar en un misma plataforma, diferentes herramientas utilizadas para los estudios de Predicción de Incendios Forestales

(figura 7). La misma sigue un patrón Master-Worker al igual que la anterior solo que, la lógica del proceso Master y de cada Worker, está compuesta por un Pipeline, en una primera versión secuencial, para la realización de las diferentes etapas de cómputo. En particular, la lógica del Worker esta compuesta por una llamada a WindNinja [9] para la generación de campos de viento a partir de un viento inicial y una segunda etapa en la cual este campo, unido a otros parametrós, son introducidos como parámetros de ejecución a la aplicación Farsite [10] para la obtención del mapa de propagación del fuego. Por último, se cácula el error de dicho mapa con relación al real y se envía el error al proceso Master.

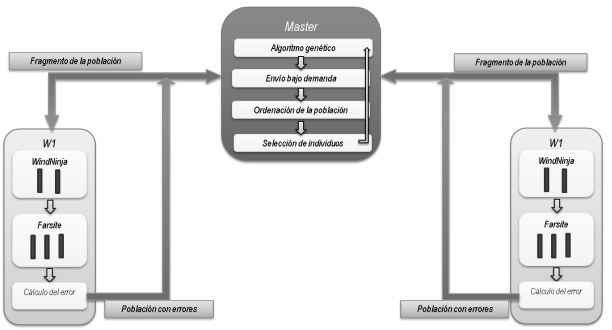

Fig. 7. Plataforma de predicción de incendios

En todos los casos, los experimentos fueron realizados en un cluster IBM xSerie con doble procesador Intel(R) Xeon(R) Dual-Core con 32 nodos. A continuación se resumen las características fundamentales del mismo:

TABLA I
CLUSTER IBM. RESUMEN DE CARACTERÍSTICAS.

32 IBM x3550 Nodes
2 x Dual-Core Intel(R) Xeon(R) CPU 5160 @ 3.00GHz 4MB L2 (2x2)
12 GB Fully Buffered DIMM 667 MHz
Hot-swap SAS Controller 160GB SATA Disk
Integrated dual Gigabit Ethernet

La figura 8 muestra el resultado de la ejecución de la aplicación de algebra matrices variando el número de threads utilizados y la cantidad de procesos involucrados para 40 iteraciones. Para unas matrices de dimensión 700x700, la ejecución más rapida se obtiene con una configuración de 4 procesos con 4 threads. A partir de este punto, el rendimiento comienza a degradarse para todos los casos. De aquí en adelante, la carga de trabajo una vez que se aumenta la cantidad de procesos, se hace lo suficientemente pequeña como para que penalice más el costo de las comunicaciones en relación a la ganancia que se obtiene con la utilización de más procesos o threads para el cómputo de la solución.

Para una carga de trabajo mayor, en este caso matrices de 3000x3000, el punto de inflexión en el rendimiento se desplaza hasta los 12 processos

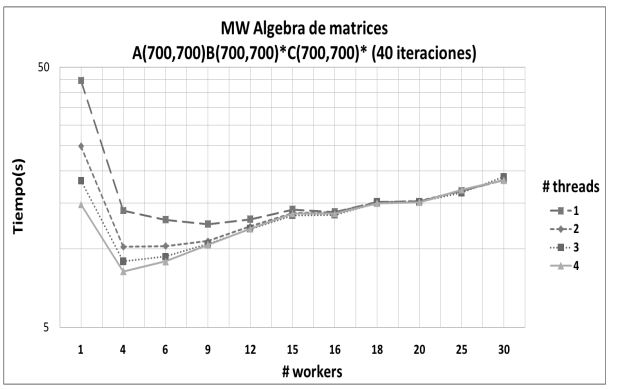

Fig. 8. Algebra de matrices. Dimensión: 700x700

MPI con 4 threads por proceso. También podemos observar que el rendimiento no mejora sustancialmente a partir de los 9 procesos con 4 threads. Esta podría ser nuestra configuración ideal de ejecución si tomaramos en cuenta en nuestra decisión, la eficiencia computacional.

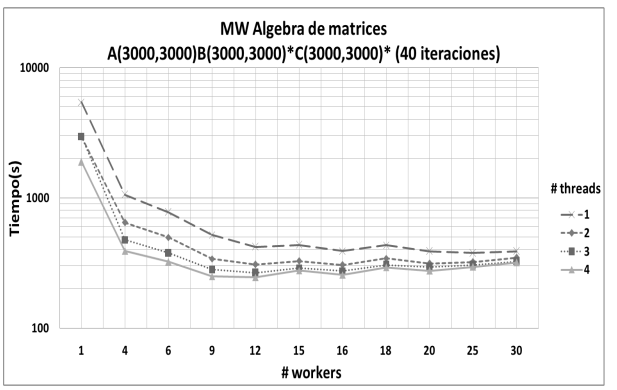

Fig. 9. Algebra de matrices. Dimensión: 3000x3000

Por las características propias del algoritmo de multiplicación de matrices y la manera en que está implementado el mismo en esta aplicación, en todos los casos, siempre se obtiene un mejor rendimiento mientras más threads por procesos MPI utilizamos. Lamentablemente, está no es una condición general, en experimentos realizados por nuestro departamento utilizando los bechmarks de NAS [11] [12] se han obtenido resultados en los cuales el mejor rendimiento se optiene con la utilización un subconjunto de los *cores* del nodo de cómputo y dicho valor varía dependiendo de la arquitectura utilizada.

Los resultados experimentales utilizando la plataforma de predicción de incendios forestales (Figura 10) muestran un resultado diferente. En este caso, el rendimiento de la aplicación mejora siempre que utilizamos un mayor número de procesos. Tambien existe una mejora del rendimiento si aumentamos la cantidad de threads utilizados en cada nodo de cómputo, aunque en menor proporción en comparación con el aumento de la cantidad de procesos.

La causa de este comportamiento puede ser explicada por dos factores fundamentales. La afectación del

rendimiento de la aplicación producto de costo de las comunicaciones es despreciable ya que la información referente a cada individuo enviado por el master a los workers esta compuesta por una estructura de 64 bytes. Por otro lado, la calidad de paralelismo de la regiones de cómputo OpenMP en la aplicación Farsite podríamos considerarla bastante buena ya que se comprueba una mejora notable en el rendimiento si aumentamos la cantidad de threads utilizados en estas regiones. Sin embargo, las mismas representan aproximandamente un 50% del tiempo de ejecución de toda la aplicación. Por lo tanto, la influencia de este código paralelo en el tiempo total de ejecución, se ve reducida por el peso significativo que tiene la región serie del código de dicha aplicación.

Un aumento en la cantidad de procesos implica que la cantidad de de individuos a procesar por los workers disminuye en proporción inversa al igual que el tiempo total de ejecución por worker. Esta relación de mejora no se obtiene con el aumento de la cantidad de threads por proceso, de ahí que para los experimentos realizados, la mejora el los tiempo de ejecución es más notable con el aumento en la cantidad de workers y no con el aumento de la cantidad de threads.

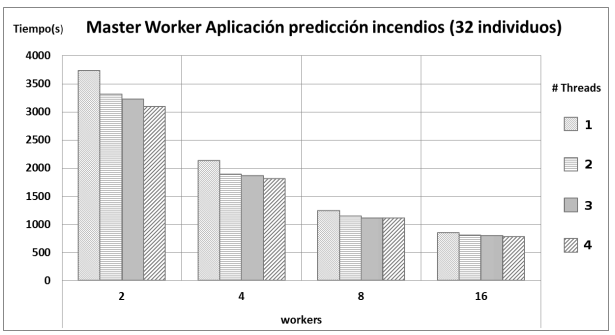

Fig. 10. Predicción de incendios forestales

Para ambas aplicaciones, no se detectaron influencias significativas en el rendimiento cuando se varía en tipo de afinidad utilizada. Este comportamiento, es producto de la forma en que está implementada la región de cómputo paralelo y aunque coincide en ambos casos, no es condición suficiente para asegurar que este factor no tenga influencia en otro tipo de aplicaciones.

IV. Conclusiones

En este artículo hemos identificado los factores de rendimiento fundamentales para las aplicaciones híbridas (MPI+OpenMP): balance de carga de trabajo, número de procesos, número de threads, afinidad, calidad del paralelismo en las regiones OpenMP, patrón de comunicación y la carga de trabajo de los threads de comunicación. En particular, para aplicaciones tipo Master-Worker y Pipeline, podemos descartar la influencia de los dos últimos factores. En el caso del patrón de comunicación, el mismo el rígido a partir de la definición del resto de los parámetros. Por este motivo que lo descartamos.

En el segundo caso, la carga de trabajo de los threads de comunicación, este factor debe tomarse en consideración solo si la lógica de la aplicación no se afecta si enviamos los mensajes MPI antes que se realice el cómputo completo de una fase de la misma. Como las aplicaciones Master-Worker y Pipeline normalmente suelen no cumplir con esta condición, no deberíamos considerarlo.

Por otra parte, hemos definido para estos dos tipos de aplicaciones, una estrategía general a seguir para ajustar los factores dependiendo del comportamiento general de la aplicación.

Nuestro trabajo futuro se concentra en concretar una expresión analítica que nos permita predecir una buena configuración de estos factores de rendimiento con el objetivo de conseguir una mejoría en los tiempos de ejecución para este tipo de aplicaciones.

Agradecimientos

El presente trabajo ha sido financiado mediante los proyectos:

Computación de Altas Prestaciones y su Aplicación a la Ciencia e Ingeniería Computacional, Referencia: TIN2007-64974.

Ejecución Eficiente de Aplicaciones Multidisciplinares: Nuevos Desafíos en la Era Multi/Many Core, Referencia: TIN2011-28689-C02-01.

Referencias

[1] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes," in *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, feb. 2009, pp. 427–436.

[2] Rolf Hempel, "The mpi standard for message passing," in *High-Performance Computing and Networking*, Wolfgang Gentzsch and Uwe Harms, Eds., vol. 797 of *Lecture Notes in Computer Science*, pp. 247–252. Springer Berlin / Heidelberg, 1994.

[3] M. Sato, "Openmp: parallel programming api for shared memory multiprocessors and on-chip multiprocessors," in *System Synthesis, 2002. 15th International Symposium on*, oct. 2002, pp. 109 –111.

[4] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *Computational Science Engineering, IEEE*, vol. 5, no. 1, pp. 46 –55, jan-mar 1998.

[5] E. Cesar, A. Moreno, J. Sorribes, and E. Luque, "Modeling master/worker applications for automatic performance tuning," *Parallel Computing*, vol. 32, no. 7-8, pp. 568–589, 2006, Algorithmic Skeletons.

[6] A. Moreno, E. Cesar, A. Guevara, J. Sorribes, and T. Margalef, "Load balancing in homogeneous pipeline based applications," *Parallel Computing*, vol. 38, no. 3, pp. 125–139, 2012.

[7] L. M. Liebrock and S. P. Goudy, "Methodology for modelling spmd hybrid parallel computation," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 8, pp. 903–940, 2008.

[8] Ronal Muresano, Dolores Rexachs, and Emilio Luque, "Methodology for efficient execution of spmd applications on multicore environments," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, may 2010, pp. 185–195.

[9] J. M. Forthofer, K. Shannon, and B. W. Butler, "Simulating diurnally driven slope winds with windninja," in *8th Symposium on Fire and Forest Meteorological Society*, 2009.

[10] M. A. Finney, *FARSITE, Fire Area Simulator–model development and evaluation*, Res. Pap. RMRS-RP-4, Ogden, UT: U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station. 1998.

[11] Abdul Waheed and Jerry Yan, "Parallelization of nas benchmarks for shared memory multiprocessors," in *High-Performance Computing and Networking*, Peter Sloot, Marian Bubak, and Bob Hertzberger, Eds., vol. 1401 of *Lecture Notes in Computer Science*, pp. 377–386. Springer Berlin / Heidelberg, 1998.

[12] Haoqiang Jin and Rob F. Van der Wijngaart, "Performance characteristics of the multi-zone nas parallel benchmarks," *Journal of Parallel and Distributed Computing*, vol. 66, no. 5, pp. 674–685, 2006, IPDPS '04 Special Issue.