Una herramienta de benchmarking para compiladores de OpenACC

Daniel Barba,[1] Arturo González-Escribano[2] y Diego R. Llanos[3]

Resumen— **OpenACC existe ya desde hace algunos años y, durante los mismos, han ido apareciendo una serie de compiladores tanto en el ámbito académico como en la industria. Debido a la novedad del estándar OpenACC así como al continuo desarrollo de los compiladores existentes, una suite de benchmarks específicamente creada para analizar el comportamiento del código generado por estos compiladores en distintas máquinas adquiere una utilidad importante. En este artículo presentamos *TORMENT OpenACC*, una suite de benchmarks preparada para ser compilada por diferentes compiladores y que ofrece un resumen de los resultados obtenidos. Así mismo, junto a esta herramienta hemos desarrollado una métrica adecuada para la puntuación de los pares compilador-máquina y que hemos denominado *Puntuación TORMENT_ACC*.**

Palabras clave— **OpenACC, compiladores, benchmarking.**

I. Introducción

OpenACC es un estándar abierto que define una serie de directivas de compilación o pragmas para ejecución de fragmentos de código en aceleradores de tipo GPU y Xeon Phi. Su objetivo es facilitar la paralelización de código secuencial en este tipo de aceleradores reduciendo el tiempo necesario tanto en la programación como en el aprendizaje [1]. La especificación se encuentra en la versión 2.5 [2].

La responsabilidad de la paralelización automática del código secuencial recae sobre los diferentes compiladores que soportan OpenACC: PGI Compiler [3], de *The Portland Group*, empresa ahora perteneciente a Nvidia, es según los estudios que hemos realizado el compilador con un grado de madurez más alto. Se encuentra disponible como parte del *Nvidia OpenACC Toolkit*, gratuito durante tres meses y con la posibilidad de adquirir licencias comerciales o de investigación. OpenUH [4], de la Universidad de Houston y accULL [5] de la Universidad de La Laguna son dos alternativas de código abierto desarrolladas en el ámbito académico y que pueden descargarse libremente para su uso.

Además de los compiladores señalados existen otros compiladores que soportan OpenACC, pero debido a que en la fecha de redacción de este artículo fue imposible obtener licencias de prueba o académicas, han sido dejados fuera de nuestros estudios. Estos compiladores están siendo desarrollados por *CRAY Inc.* y *Pathscale Inc.*.

[1]Dpto. de Informática, Universidad de Valladolid, e-mail: daniel@infor.uva.es.

[2]Dpto. de Informática, Universidad de Valladolid, e-mail: arturo@infor.uva.es.

[3]Dpto. de Informática, Universidad de Valladolid, e-mail: diego@infor.uva.es.

Nuestro trabajo consiste en el desarrollo de una herramienta de análisis de rendimiento del código generado por estos compiladores que hemos denominado *TORMENT OpenACC* (Trasgo perfORMance and EvaluatioN Tool for OpenACC). El motivo que nos ha movido a comenzar este trabajo es la escasez de herramientas similares para OpenACC y las carencias de las existentes. Estas herramientas son, a fecha de redacción de este artículo, EPCC OpenACC Benchmark Suite [6] y Rodinia [7]. El primero ha sido desarrollado por el *Edinburgh Parallel Computing Centre* y consiste en una serie de microbenchmarks, destinados a medir el overhead de la implementación de los distintos pragmas, y benchmarks para medir el rendimiento del código generado. Rodinia es una traducción de los benchmarks originales de la suite de Rodinia [8] para OpenACC desarrollado por *Pathscale*.

Nuestro trabajo busca dar respuesta a la necesidad de análisis del rendimiento y comparación del código generado por los distintos compiladores, intentando mantener un equilibrio entre las fortalezas y debilidades de los diferentes compiladores y tratando de obtener una herramienta que sea posible utilizar con los compiladores disponibles para la comunidad académica. Debido a que los compiladores se encuentran aún en fase de desarrollo, es imprescindible mantener el código lo más sencillo posible.

Además del desarrollo de la herramienta como tal, hemos desarrollado también una métrica para poder ofrecer al usuario de esta herramienta una puntuación relativa que permita la comparación entre diferentes sistemas y compiladores analizados. Esta métrica ha sido denominada *Puntuación TORMENT_ACC*

El resto del artículo se organiza del siguiente modo: La sección dos describe con más detenimiento las herramientas de benchmarking existentes. La sección tres describe los compiladores actualmente soportados por *TORMENT OpenACC*. En la sección cuatro se describe la herramienta, sus objetivos y características a medir y enumera los benchmarks actualmente implementados, así como la métrica utilizada. Finalmente, la sección cinco concluye el artículo.

II. Herramientas existentes

Como se indicaba en la introducción, a fecha de redacción de este artículo existen dos herramientas de análisis de rendimiento para OpenACC. A continuación describimos más en profundidad las mismas y las carencias detectadas.

A. EPCC OpenACC Benchmark Suite

La suite de benchmarks desarrollada por el *EPCC* ha sido diseñada específicamente para OpenACC. Se compone de tres partes diferenciadas y que denomina *Nivel 0*, *Nivel 1* y *Nivel de Aplicaciones*.

En el *Nivel 0* se pueden encontrar una serie de microbenchmarks cuyo objetivo es medir el overhead generado por las implementaciones de los pragmas. Estos microbenchmarks dan un resultado que es la diferencia de dos tiempos de ejecución en función del pragma que están analizando, o bien el tiempo de transferencia de datos en una directiva de tipo `#pragma acc data`. Estos resultados son interesantes para el desarrollo de los compiladores, pero tal y como están diseñados no ofrecen una idea clara de rendimiento y los resultados pueden ser confusos.

En el *Nivel 1* se encuentran un conjunto de benchmarks típicos de tipo BLAS basados en *Polybench* y *Polybench/GPU* [9]. Los resultados ofrecidos por estos benchmarks son tiempos de ejecución de los diferentes códigos por lo que son un buen indicador del rendimiento del código generado por los diferentes compiladores.

En el *Nivel de Aplicaciones* hay 3 benchmarks de mayor entidad que los anteriores. Estos benchmarks también ofrecen tiempos de ejecución, pero al ser problemas menos sintéticos dan resultados más interesantes.

En general, la idea del *EPCC OpenACC Benchmark Suite* está bien planteada y diseñada. No obstante, los resultados obtenidos en los microbenchmarks no son adecuados para un análisis de rendimiento. Los demás benchmarks, si bien podrían ser útiles, siempre han dado problemas en nuestros estudios. En unos casos no podían compilarse con algunos de los compiladores utilizados, en otros casos daban error de ejecución. Uno de los mayores problemas ha sido la limitación en los tamaños de los datos a utilizar debido a problemas en la implementación que llevaban a que se intentase asignar mucha más memoria de la deseada, originando errores en la ejecución y limitando a 10MB el tamaño de los datos.

B. Rodinia para OpenACC

La empresa *Pathscale Inc.* ha desarrollado una versión [7] de la suite de benchmarks Rodinia [8], [10] para su uso con compiladores de OpenACC. Con la versión existente en GitHub a día 25 de Abril de 2014, en general los benchmarks disponibles no compilan con ninguno de los compiladores existentes, salvo contadas excepciones. El compilador de PGI es el único que logra compilar un número significativo de benchmarks.

La suite se compone de benchmarks que dan el tiempo de ejecución, por lo que serían un buen punto de partida para un análisis de rendimiento. Por desgracia, la escasa madurez del código y la poca compatibilidad con los compiladores disponibles hacen imposible su uso para establecer una comparativa de rendimientos.

III. Compiladores Soportados

En la introducción hemos enumerado brevemente los compiladores disponibles. Su elección está motivada por la existencia de licencias gratuitas o académicas, o bien por ser de código abierto. A continuación explicaremos algunos detalles de los compiladores soportados en la versión preliminar de *TORMENT OpenACC*.

A. PGI Compiler

El compilador de PGI [3], desarrollado por *The Portland Group* y Nvidia, es a fecha de redacción de este artículo el compilador con un grado de madurez más alto. Su uso está muy extendido en los diferentes talleres y conferencias que se realizan sobre OpenACC. Actualmente está disponible como parte del *Nvidia OpenACC toolkit* que incluye una licencia gratuita de tres meses y la posibilidad de adquirir una licencia comercial o académica. En nuestros estudios, el compilador de PGI ha demostrado ser el más sólido y el que más alto grado de madurez tiene. Se ajusta bien a la especificación de OpenACC. Su instalación es simple y dispone de abundante documentación.

B. OpenUH

El compilador OpenUH, desarrollado por la Universidad de Houston, es una alternativa de código abierto. Comparado con el compilador de PGI, su madurez y solidez son menores y carece de algunas funcionalidades, como reducciones sobre mínimos o máximos, lo cual dificulta ligeramente la programación de aplicaciones. Su instalación, a fecha de redacción de este artículo, no es tan sencilla como cabría esperar. La versión pre-compilada disponible en su web [4] carece algunas librerías que deben ser compiladas a parte u obtenidas de otra forma.

C. accULL

El compilador accULL [5], desarrollado por la Universidad de La Laguna, es al igual que OpenUH un compilador de código abierto. De los tres compiladores soportados es el que tiene una menor solidez, teniendo problemas para la traducción fuente a fuente de algunas características de C, como punteros a función o de funcionalidades de OpenACC como algunos tipos de reducciones. La instalación de la versión disponible en su web [11] es bastante simple siguiendo las instrucciones que pueden encontrarse adjuntas al compilador en ficheros de texto.

IV. TORMENT OpenACC

Nuestra propuesta en desarrollo se denomina *TORMENT OpenACC* y es el acrónimo para Trasgo perfORMance and EvaluatioN Tool for OpenACC. El proyecto ha nacido para intentar dar una solución a la necesidad de dar respuesta a los intentos de realizar comparativas de rendimiento de código OpenACC.

A. Motivación

Las herramientas actuales han demostrado en nuestros estudios previos no ser suficientes para dar una respuesta sencilla a las comparativas de código generado por los diferentes compiladores de OpenACC. En nuestros estudios con dichas herramientas hemos encontrado una serie de problemas que estamos intentando evitar en nuestra propuesta.

El primero de los problemas es la escasa compatibilidad del código de los benchmarks entre los diferentes compiladores. Es cierto que si el código se ajusta al estándar debería compilarse y ejecutarse correctamente en cualquier implementación del estándar OpenACC, pero en la actual etapa en la que se encuentran tanto el estándar como los compiladores es muy difícil. Debido a esto hemos intentando desarrollar *TORMENT OpenACC* teniendo en cuenta tanto la especificación del estándar como las capacidades actuales de los diferentes compiladores.

Otro problema detectado es que muchos de los benchmarks no ofrecen una idea clara de rendimiento, como es el caso de los microbenchmarks del *EPCC OpenACC Benchmark Suite*. Si bien es cierto que estos microbenchmarks son útiles para el desarrollo de los compiladores, creemos que el overhead de la implementación de los distintos pragmas no es realmente relevante en una comparativa de rendimientos. Por este motivo, en nuestra propuesta dejamos fuera ese tipo de pruebas.

Finalmente, hemos observado también que, en muchos casos, benchmarks que parecen compilar correctamente luego generan errores en tiempo de ejecución o resultados incorrectos. Esta última situación puede suponer que, si no se detecta el resultado erróneo, los resultados de rendimiento obtenidos sean también incorrectos. Para evitar esto, además de incorporar comprobación de resultados a los benchmarks también analizamos los resultados durante el desarrollo de nuestra propuesta para que los benchmarks que incorporamos en nuestra herramienta compilen y ejecuten correctamente con cualquiera de los compiladores.

Otro aspecto importante que hemos tenido en cuenta a la hora de llevar a cabo el desarrollo de *TORMENT OpenACC* es la ofrecer a la comunidad una herramienta que además de facilitar el análisis de rendimiento del código generado por los compiladores de OpenACC ejecutados en distintas máquinas, ofrezcan una medida del rendimiento que permita una comparación fácil entre máquinas y compiladores. Esta idea nos ha llevado al desarrollo de la métrica *TORMENT_ACC* que describiremos posteriormente.

B. Objetivos

El objetivo principal de *TORMENT OpenACC* es la de permitir un análisis de rendimiento de código OpenACC generado por los distintos compiladores de forma sencilla, generando un resumen de resultados fácilmente analizable y dando un valor, denominado *puntuación TORMENT_ACC*, que permite la comparación de los pares máquina-compilador.

Nuestra propuesta pretende facilitar a la comunidad una herramienta preparada específicamente para OpenACC y el estado de desarrollo temprano de los compiladores existentes de forma que se eviten problemas en compilación o ejecución de otras herramientas de benchmarking existentes. *TORMENT OpenACC* estará preparado para compilarse y ejecutarse con la menor intervención posible del usuario. Recopilando la información del proceso y ofreciendo finalmente un informe en formato HTML con los datos relevantes.

Además, *TORMENT OpenACC* utilizará los compiladores GCC y NVCC para obtener datos de ejecuciones de código secuencial y código CUDA respectivamente. De este modo, nuestra propuesta ofrecerá al usuario información del *speedup* con respecto al código secuencial y CUDA ejecutado en la misma máquina.

C. Estructura de la herramienta

TORMENT OpenACC se compone de una serie de scripts que se encargan de todo el proceso de compilación, ejecución y obtención de resultados, eliminando esta carga al usuario. Estos scripts se dividen en tres categorías. Los scripts de configuración guían al usuario en la obtención de las rutas correctas a librerías CUDA (necesarias por algunos de los compiladores), rutas de compiladores y comandos de ejecución (en caso de que el usuario utilice, por ejemplo, sistemas de colas tipo slurm). El script de ejecución se encarga de la compilación y ejecución usando todos los compiladores que hayan sido hallados en el sistema del usuario. Finalmente, el script de generación de resultados procesa los resultados obtenidos y genera un fichero HTML con el informe final.

El programa propiamente dicho está desarrollado de modo que sea lo más sencillo posible para evitar problemas de compilación, evitando niveles de indirección que en otras situaciones serían aconsejables. Cada benchmark está definido en su propia unidad de compilación y está incorporado en el mismo fuente el código OpenACC y CUDA, utilizando compilación condicional y evitando tener de forma simultanea ficheros `.c` y `.cu` con código duplicado. Los ficheros `.cu` son necesarios para que el compilador NVCC genere el código CUDA correspondiente, pero esto se ha resuelto mediante el uso de enlaces simbólicos a los ficheros `.c` correspondientes.

El programa principal se encarga del lanzamiento de los benchmarks. Cada uno se lanza con 10 repeticiones con una repetición extra al comienzo cuyos resultados son desechados. Al finalizar estas diez repeticiones se obtienen los valores denominados *peak* y *average* y que corresponden a la mejor de las ejecuciones y a la media aritmética de todas ellas. Estos valores servirán para calcular la métrica de la que hablaremos posteriormente.

```
CUDA_ATTR__ int getRandom(unsigned int* seed)
{
    unsigned int next = *seed;
    int result;

    next *= 1103515245;
    next += 12345;
    result = (unsigned int) (next/65536) % 2048;

    next *= 1103515245;
    next += 12345;
    result <<= 10;
    result ^= (unsigned int) (next/65536) % 1024;

    next *= 1103515245;
    next += 12345;
    result <<= 10;
    result ^= (unsigned int) (next/65536) % 1024;

    *seed = next;
    return result;
}
```

Fig. 1
Código para la generación de números aleatorios.

D. Benchmarks implementados

La versión preliminar de *TORMENT OpenACC* contiene únicamente dos benchmarks. El desarrollo de la herramienta sigue en proceso y otros benchmarks se irán añadiendo progresivamente.

D.1 MonteCarloPi

Este benchmark consiste en una aproximación de Pi por el método de Monte Carlo, que se basa en la generación de puntos aleatorios en un cuadrado de lado unitario. Se comprueba si estos puntos se encuentran dentro de un cuarto de círculo de radio unitario y se acumula el total de puntos que cumplen dicha condición. Finalmente, se aplica la siguiente fórmula:

$$\pi \approx \frac{4*P}{T}$$

Donde P es el número de puntos dentro del cuarto de círculo y T es el total de puntos generados.

MonteCarloPi es un benchmark que no tiene prácticamente transferencias de memoria y un cálculo computacional muy simple, pero puede ser optimizado en CUDA haciendo que cada hilo calcule varios puntos y utilizando la *shared memory* de los bloques para evitar accesos a memoria global. Un buen resultado de los compiladores en este benchmark dependerá de estos factores.

Dado que no se puede hacer uso de la función srand de C en el código ejecutado en la GPU, y el uso de la librería curand se limita a código CUDA, hemos decidido replicar la función srand para permitir su ejecución en la GPU, como se indica en la fig. 1.

El código utilizado para las versiones de OpenACC y CUDA se muestra en las figs. 2 y 3

```
#pragma acc parallel loop \
    private(i, d, x, y, seed) \
    reduction(+:count)
for(i = 0; i < COORD_NUM; ++i){
    seed = 1987 ^ i*27;
    x = (double)getRandom(&seed)/(double)RAND_MAX;
    y = (double)getRandom(&seed)/(double)RAND_MAX;

    d = sqrt(x*x + y*y);
    if(d <= 1.0){
        ++count;
    }
}
```

Fig. 2
Código de MonteCarloPi para OpenACC.

```
__CUDA_GLOBAL__ void piKernel(
        const unsigned long int triesPerThread,
        unsigned long int* hits)
{
    unsigned int seed;
    int gid, tid, bid;
    int lhits;
    float x, y;
    extern __shared__ unsigned long int sdata[];

    gid = (blockIdx.x*blockDim.x) + threadIdx.x;
    tid = threadIdx.x;
    bid = blockIdx.x;
    lhits = 0;

    seed = 1987 ^ gid*27;

    for (int i = 0; i < triesPerThread; ++i){
        x = (float)getRandom(&seed)/(float)RAND_MAX;
        y = (float)getRandom(&seed)/(float)RAND_MAX;

        float d = sqrt(x*x + y*y);
        if (d <= 1.0f){
            ++lhits;
        }
    }
    sdata[tid] = lhits;
    __syncthreads();
    if (tid == 0){
        for (int i = 1; i < blockDim.x; ++i){
            lhits += sdata[i];
        }

    hits[bid] = lhits;
    }
}
```

Fig. 3
Código de MonteCarloPi para CUDA.

```
#pragma acc data copyin(S[0:sizeS],B[0:sizeB]) \
    copy(C[0:sizeB])
{
#pragma acc parallel loop private(startB) \
    firstprivate(result,sizeS,sizeB)
for (startB = 0; startB <= sizeB-sizeS; startB++){
    C[startB] = 0;
    if (startB <= result) {
        int ind;
        for(ind = 0; ind < sizeS; ind++){
            if (S[ind] != B[startB+ind]) break;
        }
        if (ind == sizeS){
            result = startB;
            C[startB] = 1;
        }
    }
}
}

result = -1;
for (startB = 0; startB <= sizeB-sizeS; startB++){
    if (C[startB] == 1){
        result = startB;
        break;
    }
}
```

Fig. 4
Código de StringMatch para OpenACC.

D.2 StringMatch

El benchmark *StringMatch* es un programa de alineamiento de cadenas de caracteres. El programa consiste en la búsqueda de la primera ocurrencia de una cadena pequeña en una cadena grande, utilizando un algoritmo *naive*. En este benchmark, la cadena grande tiene una longitud de 10 millones de caracteres, es decir 10MB. Las cadenas pequeñas, que son cuatro, tienen una longitud de 1000 caracteres o 1KB.

Este algoritmo es interesante porque combina transferencia de datos con uso eficiente de la memoria, especialmente el uso de la *shared memory*. Los códigos de las versiones de OpenACC y CUDA pueden verse en las figs. 4 y 5.

E. Métrica utilizada

Para la elección de la métrica denominada *TORMENT_ACC* hemos decidido optar por la metodología SPEC [12]. El *System Performance Evaluation Cooperative*, comúnmente conocido como SPEC, es un referente ampliamente conocido en cuanto a benchmarking y análisis de rendimiento se refiere. Una de sus fortalezas es el reconocer que los benchmarks envejecen en función del paso del tiempo y, en consecuencia, deben ser actualizados.

SPEC utiliza la siguiente metodología. En primer lugar, cada programa devuelve su tiempo de ejecución y se calcula el *SPECratio*, que consiste en el ratio obtenido de dividir un tiempo de ejecución de referencia suministrado por SPEC entre el tiempo de ejecución obtenido. Finalmente, se obtiene la media geométrica de todos los SPECratios del conjunto de

```
__global__ void kernel_busqueda(const char* b_idata,
        const int sizeB, const char* s_idata,
        const int sizeS, unsigned int* result)
{
    extern __shared__ unsigned char sdata[];
    int gid = (blockIdx.x*blockDim.x) + threadIdx.x;
    int tid = threadIdx.x;
    int sStart = THREADS_PER_BLOCK + sizeS;
    int offset;

    if (gid == 0){
        *result = -1;
    }

    __syncthreads();

    if (gid < *result){
        unsigned char* bd = &sdata[tid];
        b_idata += gid;
        unsigned char* sd = &sdata[sStart+tid];
        s_idata += tid;
        if(tid<THREADS_PER_BLOCK){
            for(offset = 0; tid+offset<sStart;
                    offset += THREADS_PER_BLOCK)
            {
                if (offset==0){
                    *bd = *b_idata;
                    bd+=THREADS_PER_BLOCK;
                    b_idata+=THREADS_PER_BLOCK;
                    continue;
                }
                *bd = *b_idata;
                bd+=THREADS_PER_BLOCK;
                b_idata+=THREADS_PER_BLOCK;
                *sd = *s_idata;
                sd+=THREADS_PER_BLOCK;
                s_idata+=THREADS_PER_BLOCK;
            }
        }

        __syncthreads();

        if (gid <= sizeB-sizeS){
            unsigned int i;
            unsigned int b = 1;
            unsigned char* sd = &sdata[sStart];
            unsigned char* bd = &sdata[tid];
            for(i = 0; b && i+15 < sizeS; i+=16){
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                b &= (*sd == *bd); sd++; bd++;
                if (!b) break;
            }
            for(; b && i < sizeS ; i++){
                b &= (*sd == *bd); sd++; bd++;
                if (!b) break;
            }
            if ( b ) atomicMin(result, gid);
        }
    }
}
```

Fig. 5
Código de StringMatch para CUDA.

benchmarks [13].

Nuestra propuesta sigue una idea similar a la de SPEC, pero con algunas variaciones. En primer lugar, la ejecución de los benchmarks que componen *TORMENT OpenACC* devuelven tres valores. *Peak Time* es el mejor tiempo de ejecución obtenido, medido en segundos. *Average Time* es el tiempo medio de todas las ejecuciones del benchmark, también en segundos. Finalmente, *Standard Deviation* es la desviación estándar del conjunto de medidas e indica la variabilidad obtenida en las ejecuciones del benchmark. Con los tiempos *peak* y *average* se calcula un ratio con respecto a los tiempos de referencia suministrados con la herramienta, y que son los tiempos de ejecución secuencial del benchmark en una máquina de referencia. Una vez ejecutados todos los benchmarks, se obtiene la media armónica de todos los ratios, tanto para *peak time* como *average time*. Estos valores que se obtienen son las puntuaciones *TORMENT_ACC peak* y *average*.

La principal diferencia entre *TORMENT OpenACC* y SPEC, a parte de la metodología de toma de tiempos de ejecución, consiste en el uso de la media armónica en lugar de la media geométrica. Esta decisión se fundamente en el hecho de que, para los objetivos de *TORMENT OpenACC* la media armónica tiene más ventajas que la media geométrica. En primer lugar, aunque la media geométrica siempre produce una ordenación consistente, no necesariamente produce la ordenación correcta [13] ya que esta media no es inversamente proporcional al tiempo de ejecución. En cambio, la media armónica si que es inversamente proporcional al tiempo de ejecución lo que hace que sea una media correcta para expresar ratios. Estas afirmaciones son compartidas por otros artículos como [14].

V. Conclusiones y Trabajo Futuro

Como conclusión, *TORMENT OpenACC* es una herramienta de análisis y comparación de rendimientos de código generado por compiladores de OpenACC, teniendo en consideración el nivel de madurez de tanto el estándar OpenACC como de los diferentes compiladores. *TORMENT OpenACC* desarrolla una suite de benchmarks específicamente diseñados para OpenACC y manteniendo la máxima portabilidad entre compiladores, permitiendo su compilación y ejecución en todos ellos.

Los resultados ofrecidos por *TORMENT OpenACC* incluyen la denominada *puntuación TORMENT_ACC*, diseñada para la comparación de pares máquina-compilador. Además, se ofrece un resumen de la ejecución de los benchmarks con una tabla de tiempos y ratios, tanto mínimos como medios e incluyendo la desviación estándar para un análisis de variabilidad de los tiempos de ejecución del código generado.

Junto con los resultados de los compiladores de OpenACC se incluyen también resultados de ejecución de código generado por los compiladores GCC y NVCC para código secuencial y CUDA. De este modo, se puede ofrecer al usuario una comparativa a nivel de máquina del rendimiento del código generado por los compiladores de OpenACC con respecto a versiones secuenciales y CUDA de los benchmarks.

El trabajo futuro a desarrollar consiste en la ampliación de la suite de benchmarks, para lograr resultados que sean verdaderamente relevantes. Tratando de cubrir los aspectos más interesantes de la ejecución de código en las GPUs. Además, una parte importante del trabajo restante consiste en mantener la compatibilidad entre compiladores y asegurar el correcto funcionamiento de la herramienta en distintas máquinas.

Agradecimientos

En memoria de Agustín de Dios Hernández.

Esta investigación ha sido parcialmente financiada por el MICINN y el programa ERDF de la Unión Europea: proyecto HomProg-HetSys (TIN2014-58876-P), la red CAPAP-H5 (TIN2014-53522-REDT) y el COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

Referencias

[1] OpenACC-standard.org, "About OpenACC," .

[2] OpenACC-standard.org, "OpenACC 2.5 draft for public comment," oct 2015.

[3] PGI, "Pgi accelerator compilers with OpenACC directives," `https://www.pgroup.com/resources/accel.htm`, nov 2015.

[4] University of Houston, "Open-source UH compiler," `http://web.cs.uh.edu/~openuh/download/`, nov 2015.

[5] Ruymán Reyes, Iván López-Rodríguez, Juan J Fumero, and Francisco de Sande, "accULL: an OpenACC implementation with CUDA and OpenCL support," in *Euro-Par 2012 Parallel Processing*, pp. 871–882. Springer, 2012.

[6] EPCC, "Epcc OpenACC benchmark suite," `https://github.com/EPCCed/epcc-openacc-benchmarks`, sep 2013.

[7] Pathscale, "Rodinia benchmark suite 2.1 with OpenACC port," `https://github.com/pathscale/rodinia`, apr 2014.

[8] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. (IISWC), 2009 IEEE International Symposium on*. IEEE, 2009, pp. 44–54.

[9] Louis-Noël Pouchet, "Polybench: The polyhedral benchmark suite," *URL: http://www. cs. ucla. edu/~ pouchet/software/polybench/[cited July,]*, 2012.

[10] Shuai Che, Jeremy W Sheaffer, Michael Boyer, Lukasz G Szafaryn, Liang Wang, and Kevin Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–11.

[11] Universidad de La Laguna, "accULL," `http://cap.pcg.ull.es/es/accULL`, nov 2015.

[12] Kaivalya M Dixit, "The spec benchmarks," *Parallel computing*, vol. 17, no. 10, pp. 1195–1209, 1991.

[13] David J Lilja, *Measuring computer performance: a practitioner's guide*, Cambridge University Press, 2005.

[14] John R Mashey, "War of the benchmark means: time for a truce," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 4, pp. 1–14, 2004.

TORMENT OpenACC Result page

Grupo Trasgo - Universidad de Valladolid

System Information

Benchmark version: 0.91
Hostname: hydra
Username: daniel

Software stack info
Kernel: Linux 3.10.0-229.4.2.el7.x86_64 x86_64
GCC: gcc (GCC) 4.8.3 20140911 (Red Hat 4.8.3-9)
NVCC: Cuda compilation tools, release 7.5, V7.5.17
PGI Compiler: pgcc 15.7-0 64-bit target on x86-64 Linux -tp haswell
OpenUH Compiler: OpenUH 3.1.0 (based on Open64 Compiler Suite: Version 5.0)
accULL Compiler: Release 0.4alpha

CPU info
Model: Intel(R) Xeon(R) CPU E5-2609 v3 @ 1.90GHz
Architecture: x86_64
Number of Cores: 6
Max MHz: MHz
Min MHz: MHz
L1 Cache: 32K
L2 Cache: 256K
L3 Cache: 15360K
RAM: 65687144 kB

GPU(s) info
GPU0: NVIDIA Corporation GK110B [GeForce GTX Titan Black] (rev a1)
GPU1: NVIDIA Corporation GK110B [GeForce GTX Titan Black] (rev a1)
GPU2: NVIDIA Corporation GK110B [GeForce GTX Titan Black] (rev a1)
GPU3: NVIDIA Corporation GK110B [GeForce GTX Titan Black] (rev a1)

GCC_Sequential Compiler Results:

GCC_Sequential					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	14.763701	0.43	14.764007	0.43	0.000336
StringMatch	20.118267	0.57	20.136980	0.57	0.021079

NVCC_CUDA Compiler Results:

NVCC_CUDA					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	0.016409	383.96	0.017840	354.36	0.001254
StringMatch	0.105644	108.48	0.105889	109.21	0.000162

PGI Compiler Results:

PGI					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	0.067226	93.72	0.077228	81.86	0.008222
StringMatch	0.688718	16.64	0.758494	15.25	0.068944

TORMENT_ACC_0.91 Peak: 28.26
Speedup vs Sequential (peak): 57.92x
Speedup vs CUDA (peak): 0.17x

TORMENT_ACC_0.91 Average: 25.71
Speedup vs Sequential (average): 52.40x
Speedup vs CUDA (average): 0.15x

OpenUH Compiler Results:

OpenUH					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	0.153154	41.14	0.165321	38.24	0.006447
StringMatch	1.574083	7.28	1.578459	7.33	0.002537

TORMENT_ACC_0.91 Peak: 12.37
Speedup vs Sequential (peak): 25.35x
Speedup vs CUDA (peak): 0.07x

TORMENT_ACC_0.91 Average: 12.30
Speedup vs Sequential (average): 25.07x
Speedup vs CUDA (average): 0.07x

accULL Compiler Results:

accULL					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	0.135701	46.43	0.139198	45.42	0.007257
StringMatch	1.173044	9.77	1.184555	9.76	0.004565

TORMENT_ACC_0.91 Peak: 16.14
Speedup vs Sequential (peak): 33.08x
Speedup vs CUDA (peak): 0.10x

TORMENT_ACC_0.91 Average: 16.07
Speedup vs Sequential (average): 32.76x
Speedup vs CUDA (average): 0.10x

Peak score uses the best of 10 results. Average score uses the arithmetic mean of 10 results. In both cases higher is better.

TORMENT OpenACC Result page

Grupo Trasgo - Universidad de Valladolid

System Information

Benchmark version: 0.91
Hostname: corikLaptop
Username: dani

Software stack info
Kernel: Linux 3.16.0-4-amd64 x86_64
GCC: gcc (Debian 4.9.2-10) 4.9.2
NVCC: Cuda compilation tools, release 7.5, V7.5.17
PGI Compiler: pgcc 15.7-0 64-bit target on x86-64 Linux -tp haswell
OpenUH Compiler: OpenUH 3.1.0 (based on Open64 Compiler Suite: Version 5.0)
accULL Compiler: Release 0.4alpha

CPU info
Model: Intel(R) Core(TM) i5-4200H CPU @ 2.80GHz
Architecture: x86_64
Number of Cores: 4
Max MHz: 3400.0000 MHz
Min MHz: 800.0000 MHz
L1 Cache: 32K
L2 Cache: 256K
L3 Cache: 3072K
RAM: 3939364 kB

GPU(s) info
GPU0: NVIDIA Corporation GM107M [GeForce GTX 850M] (rev a2)

GCC_Sequential Compiler Results:

GCC_Sequential					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	6.196389	1.02	6.196760	1.02	0.000310
StringMatch	11.292974	1.01	11.298524	1.02	0.006537

NVCC_CUDA Compiler Results:

NVCC_CUDA					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	0.057900	108.81	0.062234	101.58	0.004161
StringMatch	0.335251	34.19	0.335390	34.48	0.000085

PGI Compiler Results:

PGI					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	0.343979	18.32	0.378440	16.70	0.034409
StringMatch	1.925690	5.95	2.120944	5.45	0.192720

TORMENT_ACC_0.91 Peak: 8.98
Speedup vs Sequential (peak): 8.84x
Speedup vs CUDA (peak): 0.17x

TORMENT_ACC_0.91 Average: 8.22
Speedup vs Sequential (average): 8.05x
Speedup vs CUDA (average): 0.16x

OpenUH Compiler Results:

OpenUH					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	0.697705	9.03	0.698031	9.06	0.000198
StringMatch	2.224097	5.15	2.226121	5.19	0.003758

TORMENT_ACC_0.91 Peak: 6.56
Speedup vs Sequential (peak): 6.46x
Speedup vs CUDA (peak): 0.13x

TORMENT_ACC_0.91 Average: 6.60
Speedup vs Sequential (average): 6.46x
Speedup vs CUDA (average): 0.13x

accULL Compiler Results:

accULL					
Benchmark	**Peak Time (s)**	**Peak Ratio**	**Avg. Time (s)**	**Avg. Ratio**	**Std. Deviation**
MonteCarloPi	0.675184	9.33	0.676132	9.35	0.000704
StringMatch	2.298089	4.99	2.299703	5.03	0.003703

TORMENT_ACC_0.91 Peak: 6.50
Speedup vs Sequential (peak): 6.40x
Speedup vs CUDA (peak): 0.12x

TORMENT_ACC_0.91 Average: 6.54
Speedup vs Sequential (average): 6.40x
Speedup vs CUDA (average): 0.13x

Peak score uses the best of 10 results. Average score uses the arithmetic mean of 10 results. In both cases higher is better.