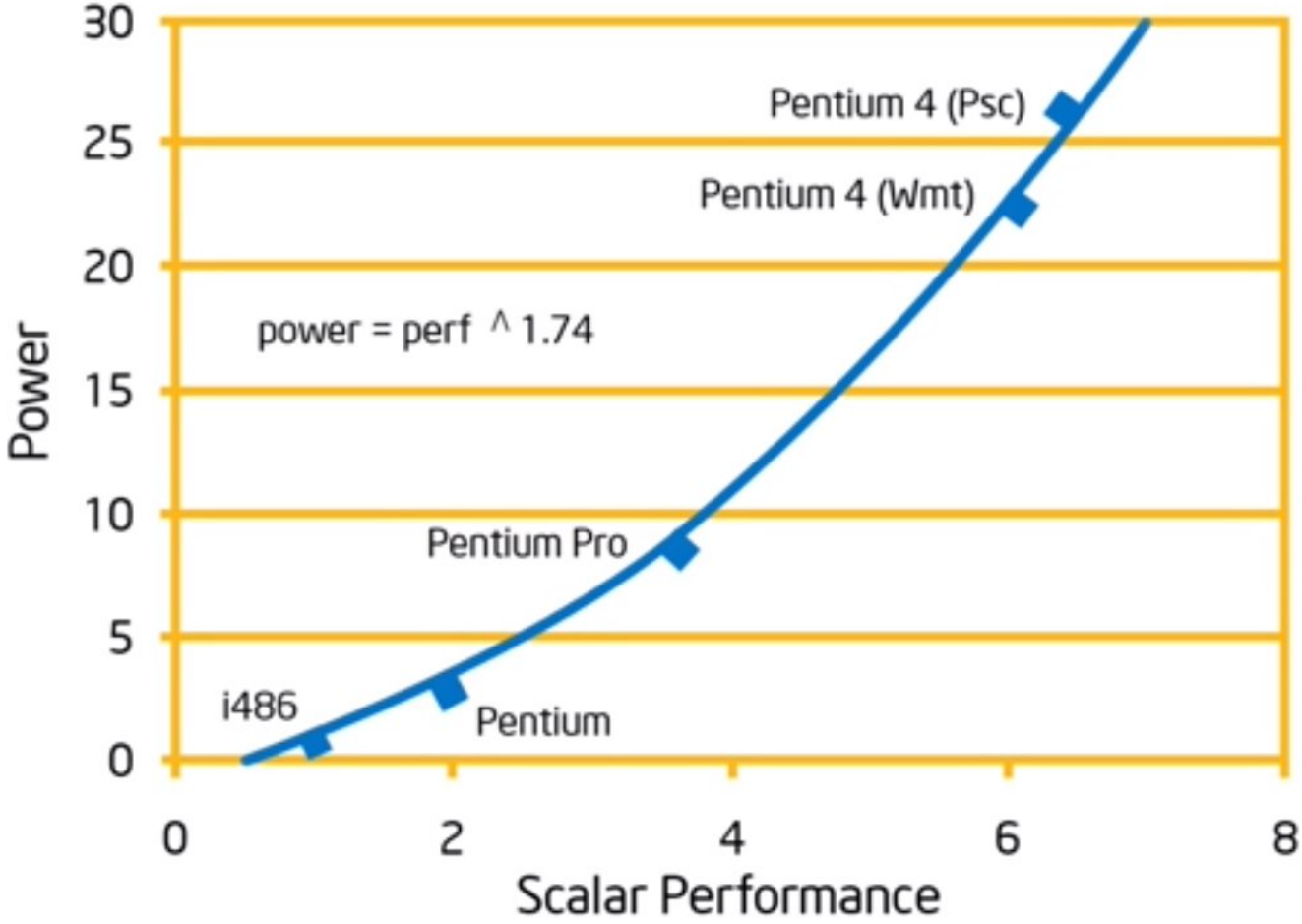
# Computación Paralela OpenMP

# *Potencia computacional*

- *Siempre se pensó que obtener mayor potencia computacional era solamente a través de hardware más potente con el mismo modelo de programación.*
- *Consideraciones energéticas han demostrado que eso no es posible, que se obtiene mayor potencia computacional con procesamiento paralelo y esto requiere programación paralela.*

# Computer Architecture and the Power Wall




Growth in Power is Unsustainable

Source: E. Grochowski of Intel

# *Potencia computacional*

- *La potencia necesaria tiene como componentes:*

  *a) la capacitancia "C", que mide la habilidad de un circuito en almacenar energía.*
  *C = q/V (carga/Voltaje), esto conduce a que q = CV*

  *b) Trabajo es mover una carga a través de una distancia, en términos electrostáticos es empujar q de 0 a V voltios.*
  *W = q * V,  por lo que* $W = CV^2$
  *W (Trabajo en Joules), q (Carga en Coulombios) V (Tensión eléctrica en Voltios). C (Capacitancia eléctrica en Faradios).*

# *Potencia computacional*

- *La potencia necesaria tiene como componentes (continuación):*

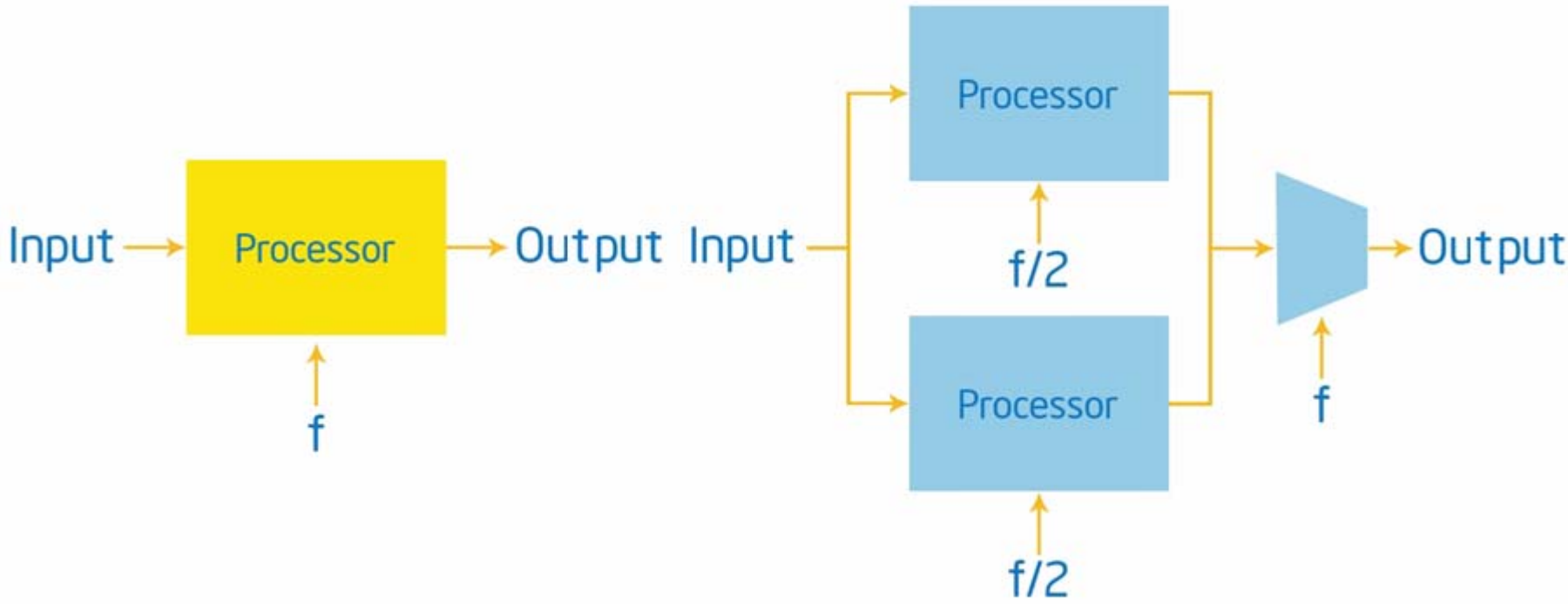
  *c) Potencia es la velocidad para realizar un trabajo, en términos eléctricos la cantidad de veces por segundo que se hace oscilar un circuito:*

  $P = W * f$

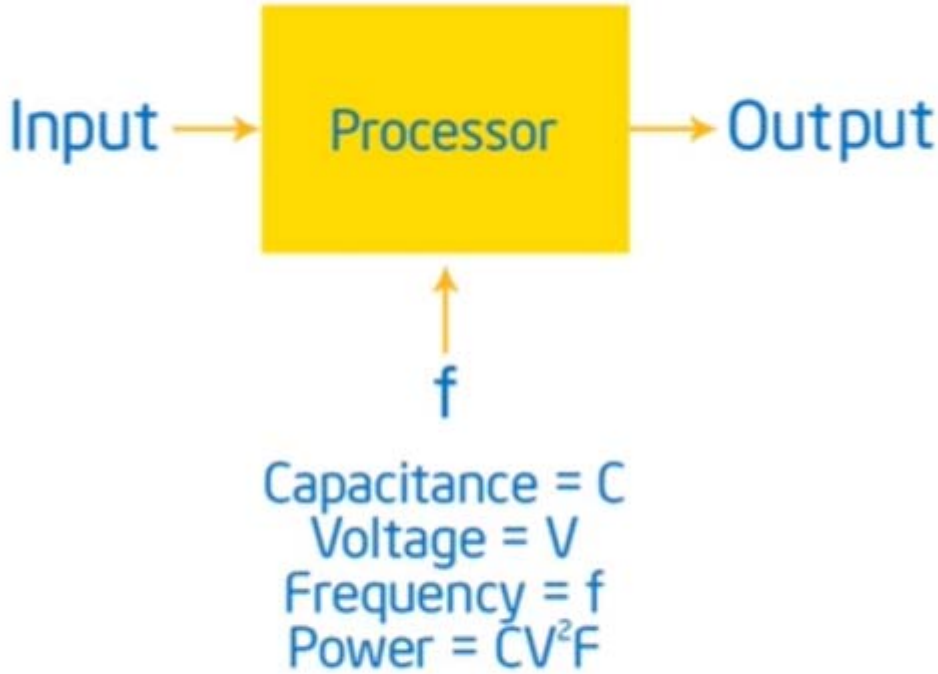
  *f (frecuencia en Hertz) P (Potencia en Watts)*

  *De aquí tenemos que* $P = CV^2f$

# *Potencia computacional*



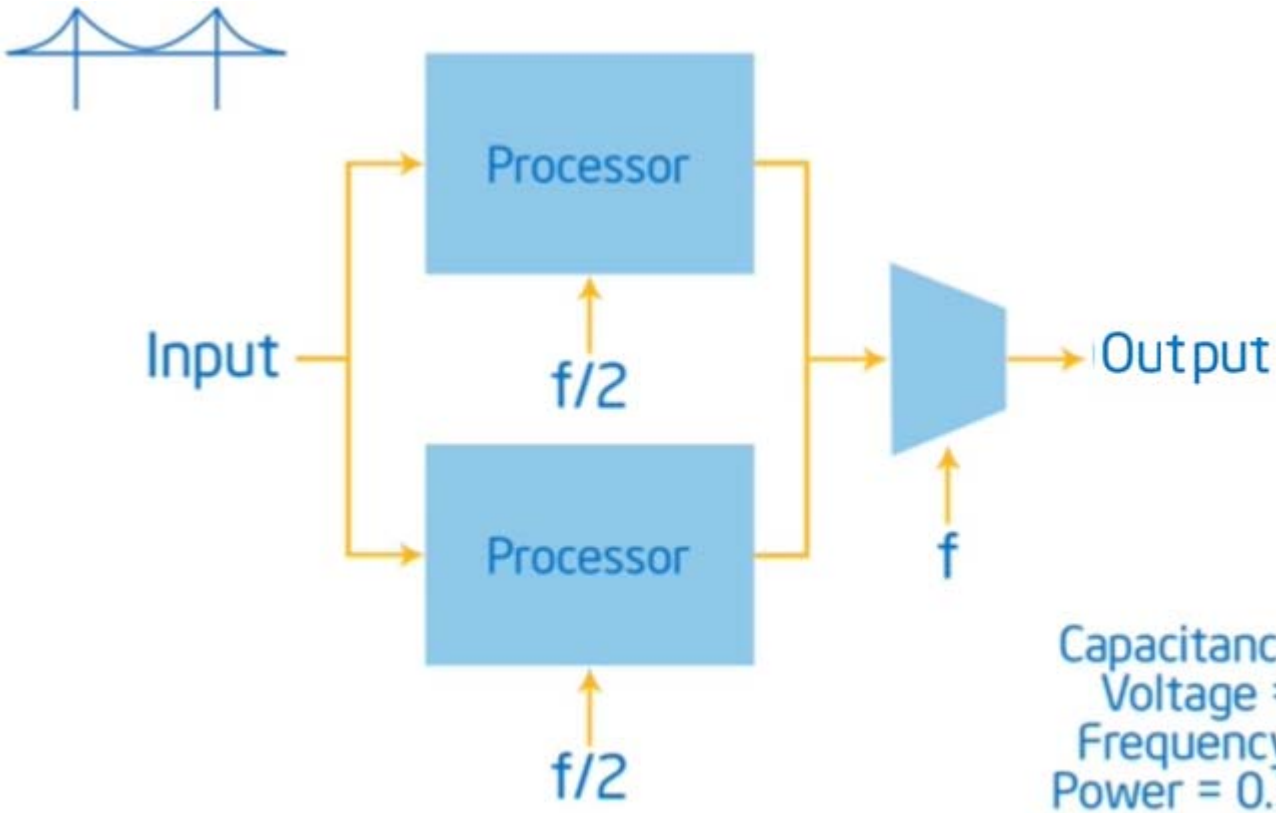
# *Potencia computacional*




Chandrakasan, A.P.; Potkonjak, M.; Mehra, R.; Rabaey, J.; Brodersen, R.W., "Optimizing power using transformations," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.14, no.1, pp.12-31, Jan 1995

Source:
Vishwani Agrawal

# *Potencia computacional*




Chandrakasan, A.P.; Potkonjak, M.; Mehra, R.; Rabaey, J.; Brodersen, R.W., "Optimizing power using transformations," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,, vol.14, no.1, pp.12-31, Jan 1995

Source: Vishwani Agrawal

# *Concurrencia vs. Paralelismo*

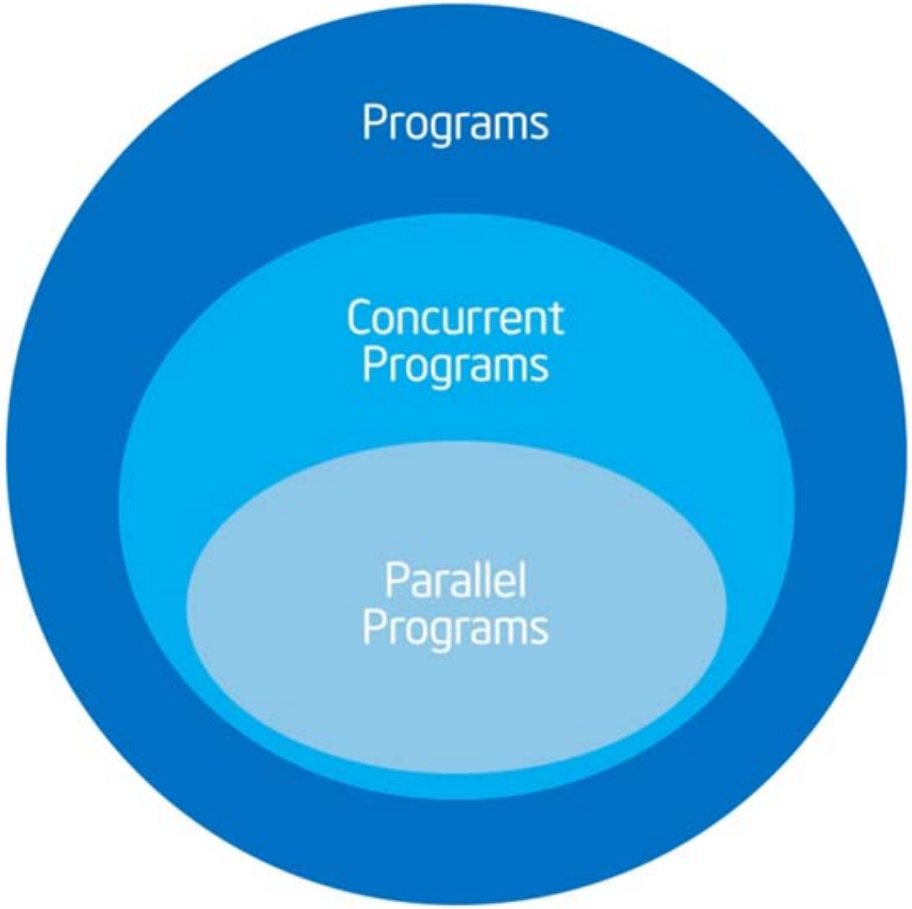
- *Concurrencia es la condición de un sistema en el cual múltiples tareas están lógicamente activas en un momento determinado.*
- *Paralelismo es la condición de un sistema en el cual múltiples tareas están físicamente activas en un momento determinado.*

# *Concurrencia vs. Paralelismo*

Concurrent, non-parallel execution

Concurrent, parallel execution

# *Programas, concurrencia y paralelismo*



# *OpenMP*

- *OpenMP es un lenguaje de programación paralela.*
- *La necesidad de computación paralela dentro de un mismo sistema proviene de una cuestión de performance vs potencia necesaria.*
- *A medida que aumentamos la performance la potencia necesaria se incrementa en forma casi exponencial.*

# *OpenMP*

- *Un procesador que trabaja a una frecuencia determinada obtiene su mejor performance usando un cierto valor de potencia.*
- *Se obtiene la misma performance con dos núcleos trabajando a mitad de frecuencia y usando aproximadamente un 40% de la potencia.*

# *OpenMP*

- *La situación se extiende al utilizar más núcleos por chip.*
- *Esto hace que el incremento de performance de los procesadores pase por ser multinúcleos.*
- *La programación de múltiples procesadores debe hacerse explícitamente para dividir la tarea entre los integrantes.*

# *OpenMP*

- *Hubo varios intentos de optimizar la programación haciendo un paralelismo automático de códigos.*
- *Prácticamente todos los intentos fracasaron.*
- *El proceso de paralelización para aprovechar múltiples núcleos por el momento debe realizarse manualmente.*

# *OpenMP*

- *Esto implica que la performance ahora tiene como componente fundamental al soft y no solamente la "fuerza bruta" del hardware.*
- *OpenMP se ha constituido en un estándar para programación paralela y nos permite trabajar con los lenguajes clásicos como C, C++ y Fortran 90.*

# OpenMP

**OpenMP** es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen el comportamiento en tiempo de ejecución.

Definido juntamente por un grupo de proveedores de hardware y de software mayores, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas para las plataformas que van desde las computadoras de escritorio hasta las supercomputadoras. Una aplicación construida con un modelo de programación paralela híbrido se puede ejecutar en un cluster de computadoras utilizando ambos OpenMP y MPI, o más transparentemente a través de las extensiones de OpenMP para los sistemas de memoria distribuida.



## Modelo de ejecución [editar]

**OpenMP** se basa en el modelo **fork-join**, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en **K** hilos (fork) con menor peso, para luego "recolectar" sus resultados al final y unirlos en un solo resultado (join).

## Sintaxis básica [editar]

La sintaxis básica que nos encontramos en una directiva de OpenMP es para C/C++:

```
# pragma omp <directiva> [cláusula [ , ...] ...]
```

Posted on April 13, 2012

»OpenMP is Being Improved for Accelerators, Multicore and Embedded Systems

## OpenMP aims to provide parallel language support for automotive, aeronautics, biotech, and financial applications.

CHAMPAIGN, Illinois, March 27, 2012 - OpenMP, the de-facto standard for parallel programming on shared memory systems, continues to extend its reach beyond pure HPC to include embedded systems, multicore and real time systems. A new version is being developed that will include **support for accelerators, error handling, thread affinity, tasking extensions and Fortran 2003**. The OpenMP consortium welcomes feedback from all interested parties and will use this feedback to improve the next version of OpenMP.

OpenMP aims to provide high-level parallel language support for a **wide range of applications**, from automotive and aeronautics to biotech, automation, robotics and financial analysis.

The key features that the OpenMP consortium is working on include:

- **Support for accelerators**. A mechanism will be provided to describe regions of code where data and/or computation should be moved to any of a wide variety of computing devices. User experiences with the OpenACC directives will provide important information to the OpenMP effort. The minimum feature core required for an initial release has been defined.
- **Error handling**. Error handling capabilities of OpenMP will be defined to improve the resiliency and stability of OpenMP applications in the presence of system-level, runtime-level, and user-defined errors. Features to cleanly abort parallel OpenMP execution have been defined, based on conditional cancellation and user-defined cancellation points.
- **Thread affinity**. Users will be given a way to define where to execute OpenMP threads. Platform-specific data and algorithm-specific properties are separated, offering a deterministic behavior and simplicity in use. The advantages for the user are better locality, less false sharing and more memory bandwidth.
- **Tasking extensions**. The new tasking extensions being considered are task deep synchronization, dependent tasks, reduction support for tasks, and task-only threads. Task-only threads are threads that do not take part in worksharing constructs, but just wait for tasks to be executed.
- **Support for Fortran 2003**. The Fortran 2003 standard adds many modern computer language features. Having these features in the specification allows users to take advantage of using OpenMP directives to parallelize Fortran 2003 complying programs. This includes interoperability of Fortran and C, which is one of the most popular features in Fortran 2003.


»*Using OpenMP* -- the book
»*Using OpenMP* -- the examples
»*Using OpenMP* -- the forum
»Wikipedia
»OpenMP Tutorial
»More Resources

### Discuss

»User Forum
*Ask the experts and get answers to questions about OpenMP*

### Recent News

- IWOMP 2012 Registration Now Open
- OpenMP is Being Improved for Accelerators, Multicore and Embedded Systems
- OpenMP at Multicore Developers Conference 2012
- PPCES 2012: Parallel Programming in Computational Engineering and Science
- Bjarne Stroustrup Will Give Keynote at the International Workshop

OpenMP
Enabling HPC since 1997

Home | Specifications | Community | Resources | News & Events | About

# Specifications

Home > Specifications

## OpenMP 4.5 Specifications

- OpenMP 4.5 Complete Specifications (Nov 2015) *pdf*
  - OpenMP 4.5 Discussion Forum
- OpenMP 4.5 Summary Card – C/C++ (Nov 2015) *pdf*
- OpenMP 4.5 Summary Card – Fortran (Nov 2015) *pdf*
- OpenMP 4.5 Examples (Nov 2016) *pdf*
  - OpenMP 4.5 Examples Discussion Forum

## OpenMP 4.0 Specifications

- OpenMP 4.0 Complete Specifications (July 2013) (PDF)
- OpenMP 4.0 Discussion Forum
- OpenMP 4.0 Summary Card – C/C++ (October 2013 PDF)
- OpenMP 4.0 Summary Card – Fortran (October 2013 PDF)
- OpenMP Examples 4.0.2 (March 2015 PDF)
- OpenMP 4.0.1 Examples (February 2014 PDF)

# OpenMP® Technical Report 4: Version 5.0 Preview 1

This Technical Report augments the OpenMP API Specification, version 4.5, with language features for task reductions, defines a runtime interface for performance and correctness tools (OMPT), extensions to the target constructs, and contains several clarifications and fixes.

All members of the OpenMP Language Working Group

November 10, 2016

Expires November 9, 2018

We actively solicit comments. Please provide feedback on this document either to the Editor directly or in the OpenMP Forum at openmp.org

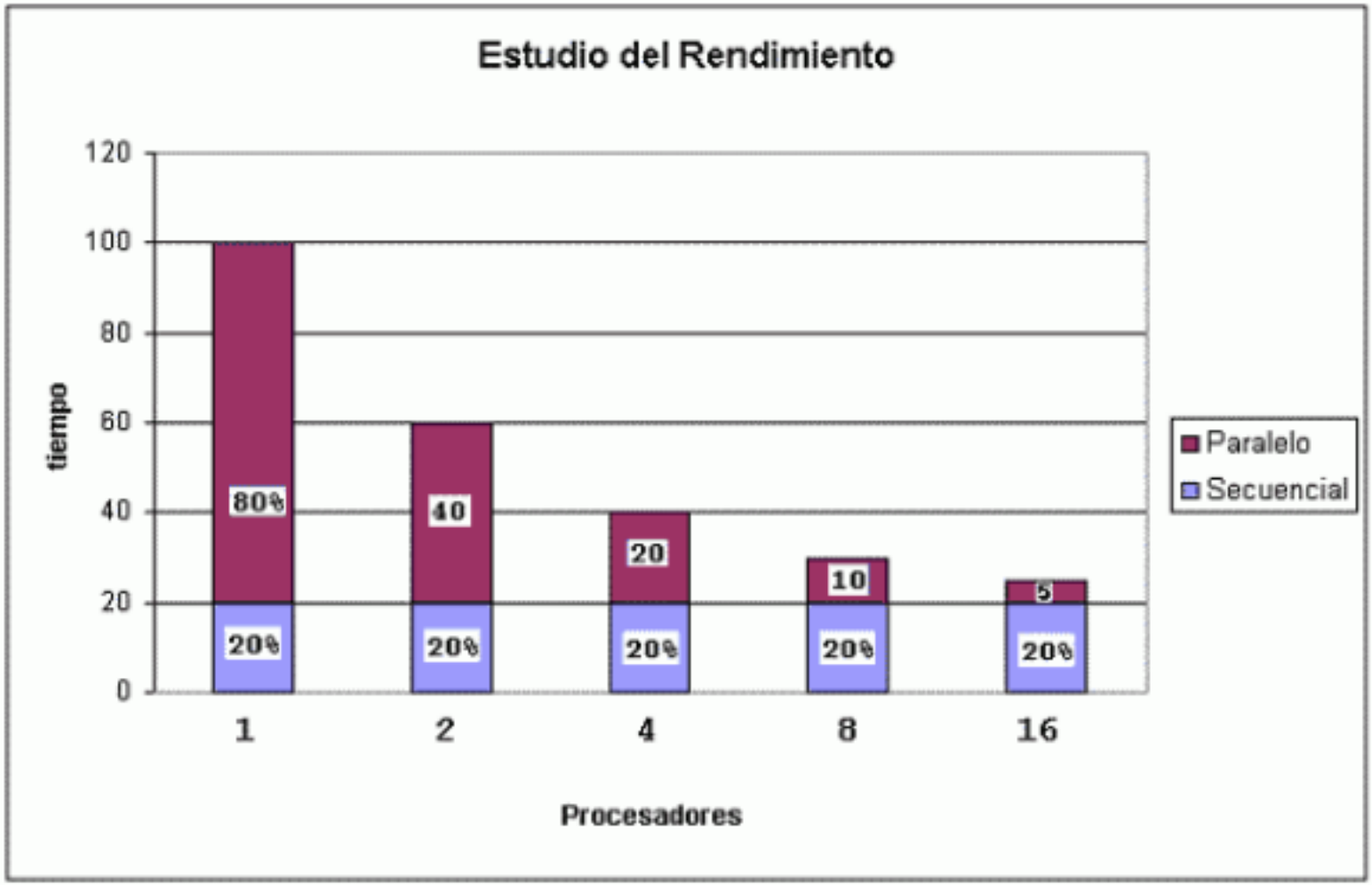
**End of Public Comment Period: January 9, 2017**

# *Rendimiento de paralelización*

*Sea un programa que posee un tiempo de ejecución de 100 unidades de tiempo.*

*El 80% de su código es perfecta y absolutamente paralelizable. Se pide calcular el Rendimiento de este programa cuando se está ejecutando sobre 1, 2, 4, 8 y 16 procesadores.*

# *Rendimiento de paralelización*



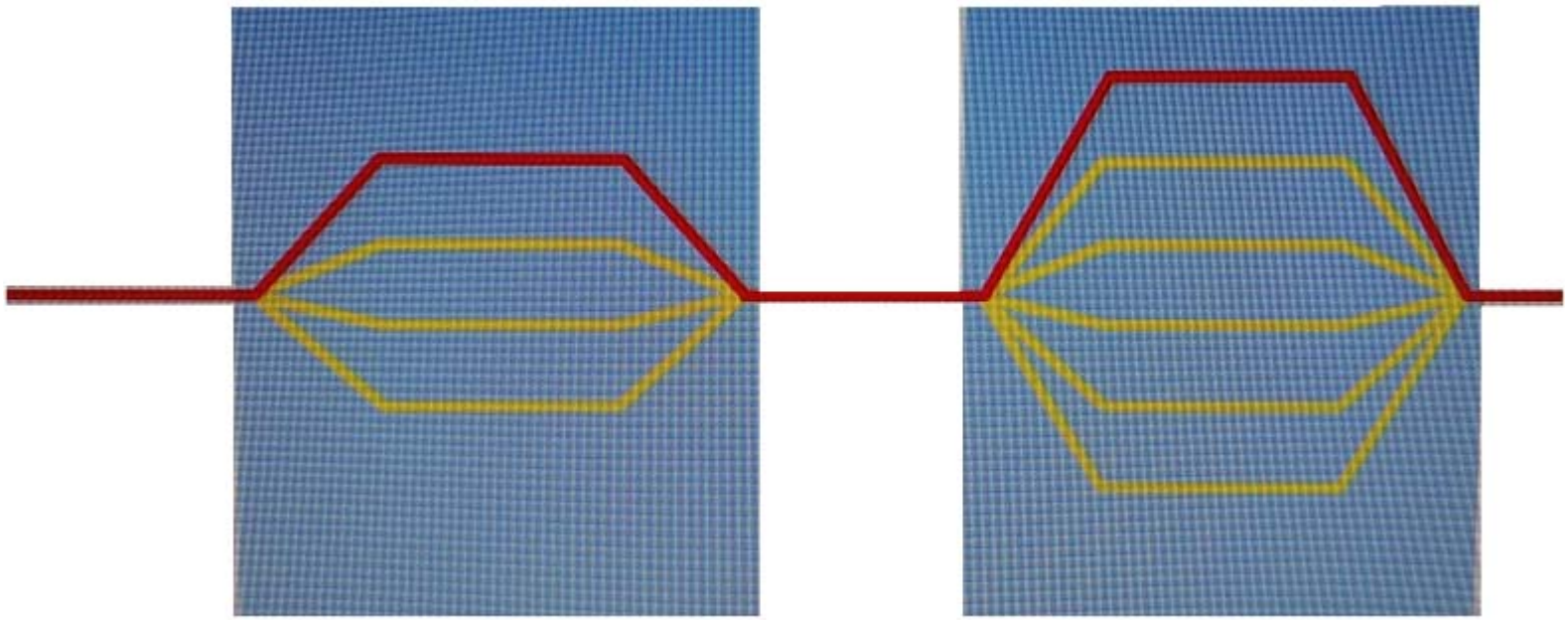
# *OpenMP*

- *OpenMP es un modelo de programación con múltiples hilos que utiliza direcciones compartidas y propias.*
- *Los hilos se comunican por el compartimiento de variables.*
- *Todos los hilos ven el montón (heap) como una área de memoria común.*
- *Para que el trabajo de hilos en paralelo sea útil, se necesita sincronización.*

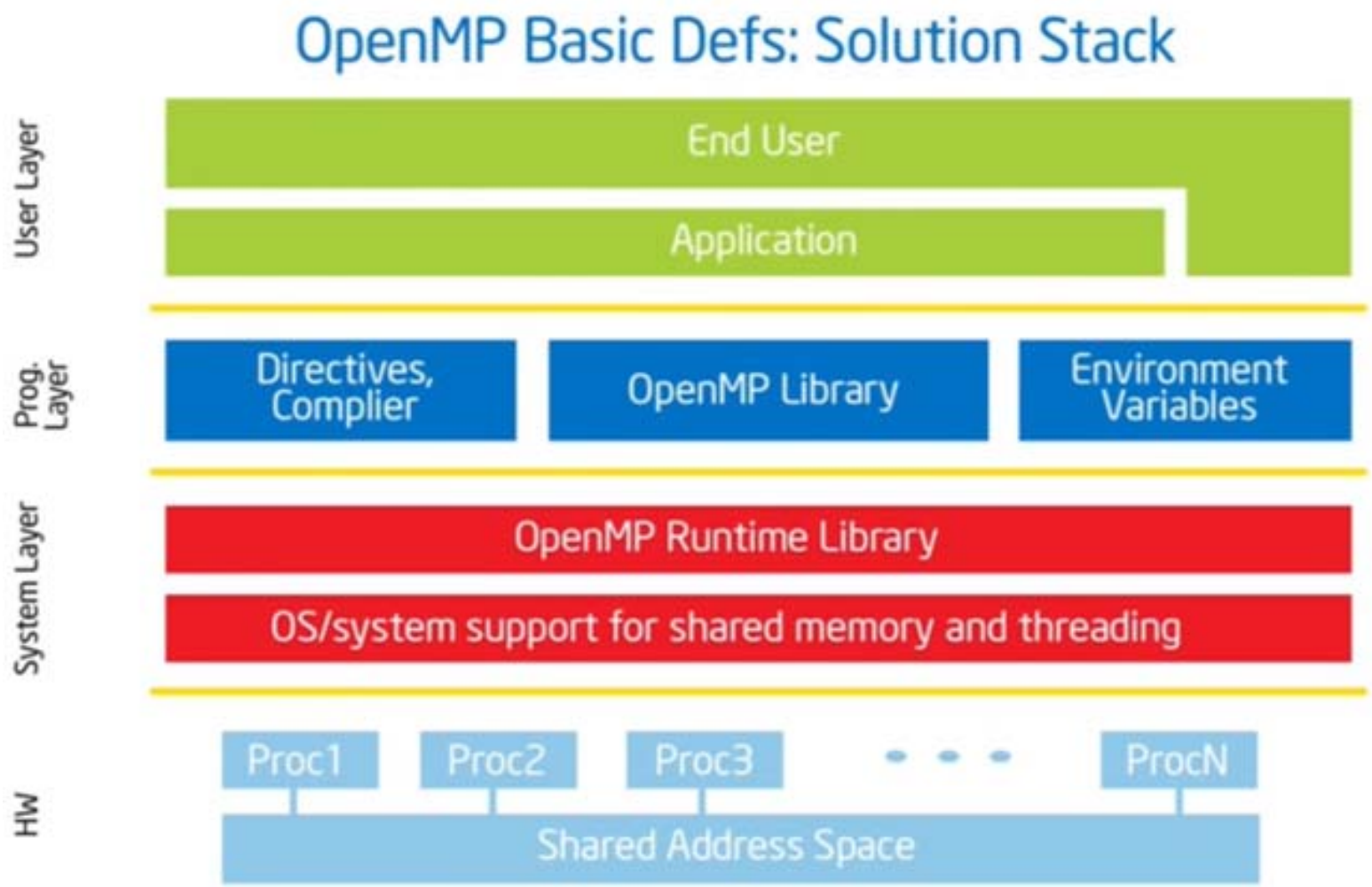
# *OpenMP*

- *Forma general:*
  **`#pragma omp construct [clause[clause]...]`**
- *Ejemplo:*
  **`#pragma omp parallel num_threads(4)`**
- *La mayoría de las construcciones OpenMP se aplican a un "bloque estructurado", esto es, un conjunto de sentencias con un punto de entrada al principio y un punto de salida al final.*

# *Modelo de trabajo: fork-join*



# *Stack de OpenMP*



# *OpenMP*

- *Compilación y enlazado:*

  *Los programas OpenMP deben incluir la biblioteca (en C):*

  ```
  #include <omp.h>
  ```

- *El enlazado debe hacerse asi:*

  ```
  gcc -o programa programa.c -fopenmp
  ```

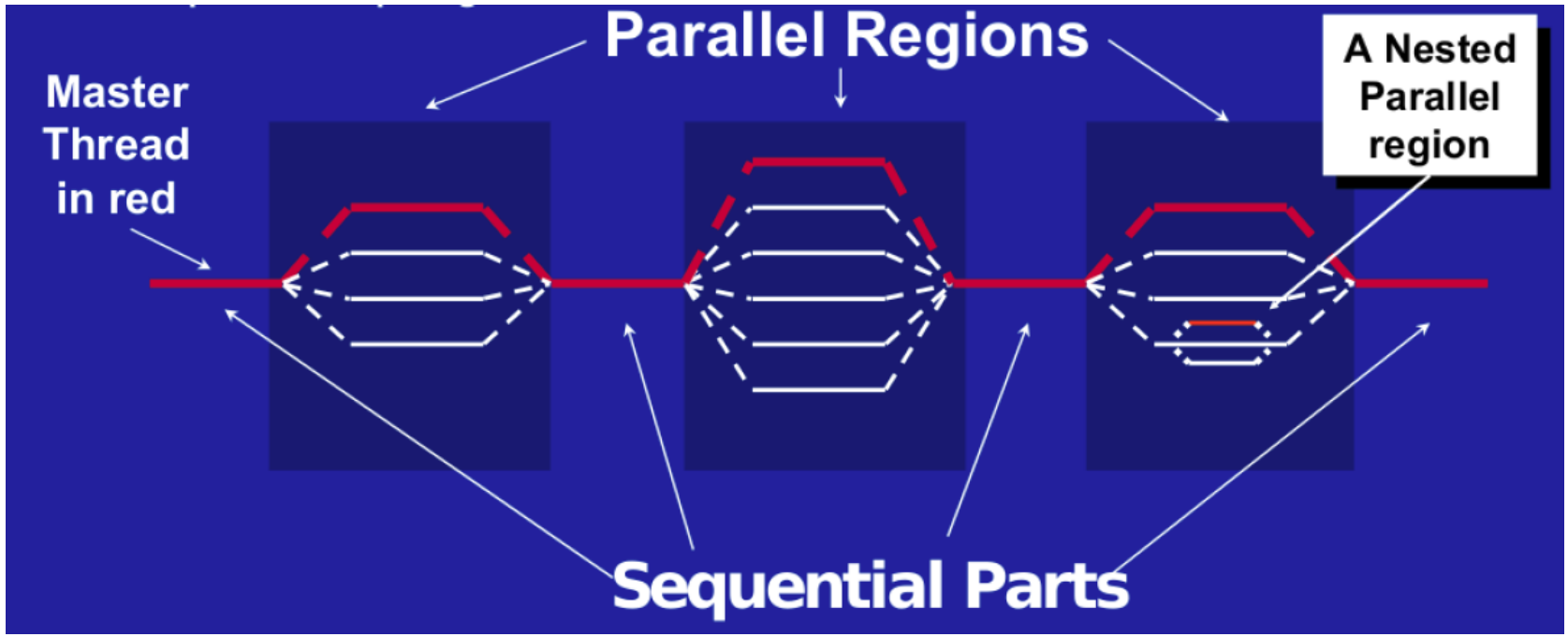
- *El ejecutable se corre como es habitual sin necesidad de elementos adicionales.*

# *OpenMP*

- *El modelo es multihilo en memoria compartida.*
- *Si se comparte la memoria en forma no controlada se pueden generar "condiciones de carrera" y hay que usar sincronización para evitar estas situaciones.*
- *Hay un hilo maestro que convoca al resto de los hilos.*

# *OpenMP*

- *La construcción "parallel" se usa para convocar los hilos en el proceso de "fork".*



# *OpenMP - Sincronización*

- *Se puede hacer en alto o bajo nivel.*
- *En alto nivel:*
  - *critical*
  - *atomic*
  - *barrier*
  - *ordered*
- *En bajo nivel:*
  - *flush*
  - *locks*

# *OpenMP - Sincronización*

- *Critical*
- *La instrucción va a ser ejecutada en mutex.*

```
#pragma omp parallel
{
    funciones_no_criticas();
    funcion_no_critica2();
    #pragma omp critical
        funcion_critica();
}
```

# *OpenMP - Sincronización*

- *Atomic*
- *La instrucción va a ser ejecutada en mutex pero para actualizar una sola posición de memoria.*

```
#pragma omp atomic
    prod *= resultado();
```

# *OpenMP - Sincronización*

- *Barrier*
- *Los hilos esperan que todos hayan alcanzado este punto para seguir.*

```
#pragma omp parallel
{
    codigo_a_ejecutar();
    #pragma omp barrier
    /* todos se detienen en este punto */
    otro_codigo_a_ajecutar();
}
```

# *OpenMP - Sincronización*

- *Ordered*
- *El orden en que se ejecutan los hilos no está especificado y depende de las condiciones en tiempo de corrida.*
- *Es posible que ciertos eventos se ejecuten de forma ordenada con ordered, como se observa en el código siguiente, la compresión de archivos es sin orden específico, pero el envío si lo tiene.*

# *OpenMP - Sincronización*

- *Ordered*

```
#pragma omp parallel
{
    for( int n=0; n<100; n++ ) {
        compress( files[n] );
        #pragma omp ordered
            send( files[n] );
    }
}
```