Instituto Tecnológico de Costa Rica
Escuela de Computación

Esteban Meneses, PhD

Diseño de una infraestructura de computación de alto rendimiento para objetos paralelos en un lenguaje de alto nivel

Informe Final

2017

Tabla de Contenido

Datos del proyecto

Código: 540213700005

Título: *Diseño de una infraestructura de computación de alto rendimiento para objetos paralelos en un lenguaje de alto nivel*

Autores y direcciones

Coordinador: Dr. Esteban Meneses, Escuela de Computación (esmeneses@tec.ac.cr)

Resumen

La computación paralela ha alcanzado una posición predominante en la última década gracias a la abundancia de arquitecturas computacionales de múltiples núcleos. Explotar el poder computacional disponible en los sistemas modernos ofrece una enorme posibilidad de avanzar el estado del arte en la ciencia y la ingeniería. El modelo de programación de objetos paralelos ofrece muchas ventajas con respecto a otros modelos en computación paralela. Sin embargo, este modelo no ha sido explorado en el contexto de lenguajes de alto nivel. Este proyecto se enfocó en explorar las posibilidades de diseño de un sistema de computación de alto rendimiento para objetos paralelos en un lenguaje de alto nivel. Para lograr ese objetivo se hizo una recolección exhaustiva de herramientas en el lenguaje Python para computación de alto rendimiento. Esa colección demostró la oportunidad que existe al combinar los dos dominios: objetos paralelos y un lenguaje de alto nivel. Además, el proyecto creó un panorama de las posibilidades de diseño de tal combinación.

Palabras clave

Computación paralela; Objetos paralelos; Computación científica

Introducción

La industria de los semiconductores ha mantenido válida la Ley de Moore (que establece que la cantidad de transistores por área se duplica cada 12-24 meses) por más de 50 años. El resultado de esa gesta, aunado con la incapacidad de acelerar la frecuencia en los procesadores, ha propiciado la multitud de procesadores multi-núcleo que tenemos a disposición en la actualidad. Desde dispositivos móviles hasta supercomputadoras, el paralelismo es ya la forma natural de diseñar procesadores. Explotar ese paralelismo es entonces una gran oportunidad de acelerar el descubrimiento científico, el desarrollo de negocios, la formulación de políticas públicas y el mejoramiento de la calidad de vida en general.

El paradigma de programación imperante para computación paralela de alto rendimiento se basa en primitivas de paso de mensajes. Este paradigma ha demostrado ser simple, portable, de alto desempeño y escalable. Sin embargo, es de bajo nivel, lo que implica que el programador tiene que ser consciente de varios detalles de la arquitectura del sistema. En contraposición, el paradigma de objetos paralelos ha demostrado tener las mismas características y, adicionalmente, oculta los detalles de bajo nivel. El paradigma de objetos paralelos se ha implementado en lenguajes de programación como Fortran y C/C++, que no son lenguajes expresivos. Estos lenguajes no suelen ser fácilmente adoptados por científicos e ingenieros que prefieren lenguajes más expresivos de alto nivel, como Python. Junto con Fortran y C/C++, Python es considerado un lenguaje de computación de alto desempeño. Implementar el modelo de objetos paralelos en Python representa una oportunidad única de ofrecer un paradigma de programación de alto nivel junto con un lenguaje de programación expresivo.

El proyecto de investigación consistió en diseñar un sistema de objetos paralelos en Python. El proyecto comenzó con una exploración de las diferentes alternativas de lograr una aleación entre los objetos paralelos y Python. El entregable final era un diseño de la arquitectura de software necesaria junto con un prototipo del sistema. Los componentes más importantes que se deben diseñar aparecen en la Figura 1. Un sistema de tiempo de corrida debe manejar la ubicación de los objetos en los procesadores de la máquina paralela. También, el sistema debe manejar las diferentes operaciones de comunicación entre los objetos y debe proveer funcionalidades para el balanceo de carga entre los procesadores del sistema. Además, el diseño incluye una interfaz natural que permita usar los objetos paralelos en Python.

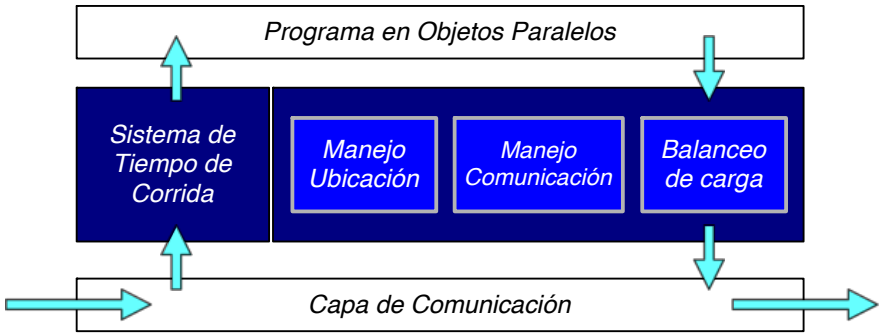

Figura 1. Vista General del Sistema de Objetos Paralelos.

Un sistema de objetos paralelos en Python tiene dos usos fundamentales. Primero, puede ser utilizada como herramienta educacional para entrenar a los programadores a razonar en computación paralela con un paradigma de alto nivel junto con un lenguaje de programación expresivo. Esta combinación tiene una barrera de aprendizaje baja comparada con las soluciones actuales. Segundo, el sistema puede ser utilizado para crear prototipos de programas en forma rápida. Así, se pueden estimar los beneficios y limitaciones de estrategias algorítmicas para resolver problemas.

El proyecto tiene como objetivo principal crear el diseño para una herramienta que pueda eventualmente facilitar el desarrollo científico y tecnológico, pilares del Instituto Tecnológico de Costa Rica. Así mismo, el proyecto ofrece una herramienta que pueda democratizar el uso de plataformas de computación de alto rendimiento para comunidades científicas e ingenieriles que no hayan sido expuestas a programación de bajo nivel.

Marco Teórico

La computación de alto rendimiento o HPC por sus siglas en inglés (*high performance computing*) tiene como objetivo utilizar la gran cantidad de procesamiento disponible en las diferentes arquitecturas de computación para resolver problemas científicos e ingenieriles [HAG10, KIR12]. Usualmente, la comunidad de HPC utiliza supercomputadoras para correr programas que representan simulaciones de modelos científicos o ingenieriles. Las supercomputadoras son equipos paralelos que aglomeran gran cantidad de nodos computacionales. Los nodos aportan todo el poder computacional, pero requieren de otros componentes, como la red de interconexión, el sistema de almacenamiento y toda la colección de software necesaria para el correcto funcionamiento de las simulaciones.

El modelo de programación más popular de los programas que corren en las supercomputadoras se denomina SPMD [LIN08] por sus siglas en inglés (*single program multiple data*). En este modelo, un mismo programa se corre en múltiples instancias sobre los nodos de la supercomputadora y las instancias intercambian información gracias al paso de mensajes. El estándar MPI (*message passing interface*) [MPI] provee una interfaz que permite programar simulaciones usando el modelo SPMD. MPI tiene varias ventajas que lo han hecho el estándar *de facto* para la programación paralela de alto rendimiento [PAC11, QUI03]. Primero, MPI es simple, ya que las primitivas en las que se basa el modelo son las de enviar y recibir un mensaje. Esas primitivas conforman el fundamento de programación de redes de computadoras y son conocidas por todos los profesionales en el área. Segundo, MPI es general, porque el estándar se puede implementar en cualquier arquitectura paralela, sin importar si la arquitectura tiene una red o no. Tercero, MPI es de alto desempeño dado que la implementación de las primitivas está muy cerca del nivel de máquina, entonces no crea una sobrecarga significativa. Cuarto, MPI es escalable, lo que significa que el mismo programa MPI puede ser corrido con un número variable de procesadores. En muchos casos, el desempeño no se deteriora aún al incrementar sustancialmente el número de procesadores [GRA03]. La Figura 2 presenta un código ejemplo en MPI.

```
#include <mpi.h>

int main(int argc, char** argv) {
    MPI_Init(NULL, NULL);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);
    printf("Hello world from processor %s, rank %d"
           " out of %d processors\n",
           processor_name, world_rank, world_size);
    MPI_Finalize();
}
```

Figura 2. Código ejemplo del lenguaje de programación MPI.

El gran inconveniente de MPI es que es un modelo de programación de bajo nivel. Eso supone que el programador debe estar consciente de varios aspectos relacionados con la arquitectura del sistema y que son independientes de la lógica del programa. Por ejemplo, el programador debe conocer cuál es la ubicación física de cualquier dato en el programa. Para ofrecer un ambiente de programación paralela productivo se deben abstraer esos detalles arquitecturales.

Un modelo de programación paralela que ofrece la bondad de abstraer esos detalles se denomina *objetos paralelos*. Este modelo conoce varias implementaciones, Charm++[CHARM++], Global Arrays [GA], ParalleX [PARALLEX], entre otros. En el modelo de objetos paralelos, el programador descompone la aplicación en objetos que interactúan entre sí. Cada objeto posee una porción de los datos y se encarga de ejecutar una función específica. Este modelo extiende la programación orientada a objetos al mundo de la computación paralela. La colección de objetos de una aplicación corresponde a las entidades que conforman la solución computacional al problema científico o ingenieril. De esta forma, existe la posibilidad de crear en el programa una ontología que recrea los agentes del problema y sus interacciones. La ubicación de los objetos es responsabilidad del sistema de tiempo de corrida y el programador puede simplemente invocar métodos en los objetos. Existe siempre para el programador una vista global de la colección del objetos en la aplicación. Los objetos exportan una interfaz de métodos que pueden ser invocados remotamente y la ejecución del programa se dice que es dirigida por mensajes.

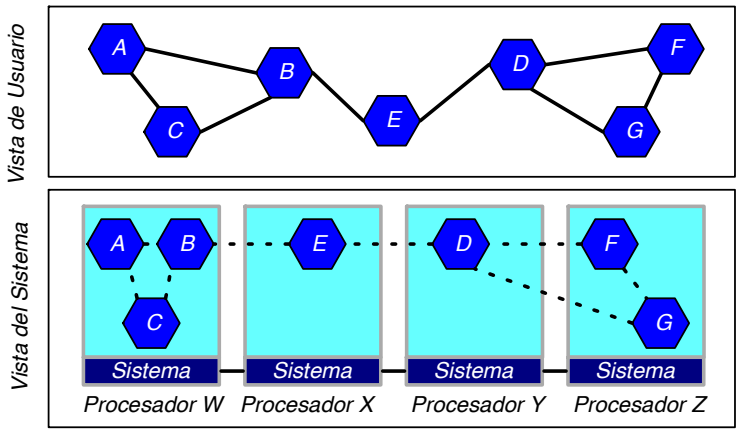

Figura 3. Modelo de Programación de Objetos Paralelos.

El modelo de objetos paralelos crea una división de responsabilidades en el sistema de computación paralelo. Esa división se manifiesta en las vistas de la Figura 3. La vista del usuario o programador ofrece una perspectiva de los objetos de la simulación científica y de las relaciones entre los objetos. La vista del sistema, por otro lado, presenta la implementación de los objetos en el sistema y la ubicación de los mismos en diferentes nodos de procesamiento. El sistema de tiempo de corrida es capaz de cambiar dinámicamente esta distribución de objetos. Inclusive, el sistema mismo puede crecer con más procesadores, o bien comprimirse al perder procesadores.

```
/******************ARCHIVO INTERFAZ*****************/
module hello {
  array [1D] Hello {
    entry Hello();
    entry void sayHi(int);
  };
};

/******************ARCHIVO ENCABEZADO*****************/
class Hello : public CBase_Hello {
public:
  Hello();
  void sayHi(int from);
};

/******************ARCHIVO FUENTE*****************/
#include "hello.decl.h"
#include "hello.h"
extern CProxy_Main mainProxy;
extern int numElements;
Hello::Hello() {}

void Hello::sayHi(int from) {
  CkPrintf("Hello from chare # %d on processor %d (told by %d).
\n", thisIndex, CkMyPe(), from);
  if (thisIndex < (numElements - 1))
    thisProxy[thisIndex + 1].sayHi(thisIndex);
  else
    mainProxy.done();
}
#include "hello.def.h"
```

Figura 4. Código ejemplo del lenguaje de programación Charm++.

La Figura 4 presenta un ejemplo de código escrito en el lenguaje Charm++, que implementa en C++ objetos paralelos. El sistema de Charm++ requiere que el programador defina un archivo de interfaz, donde se detallan los métodos remotos de los objetos (calificados por la palabra reservada *entry*). Además de ese archivo, se deben definir los archivos clásicos de C++: encabezado y fuente. El ejemplo presenta una característica negativa de Charm++ y es la complejidad del código, que es poco expresivo. La pregunta de investigación del proyecto es determinar las ventajas y desventajas de un diseño que permita tener objetos paralelos en un lenguaje de alto nivel como Python.

Metodología

Recolectar las propuestas relacionadas en la literatura. Se procederá a revisar la literatura en el tema de computación de alto desempeño y sistemas de arreglos distribuidos en Python para construir un panorama de las soluciones disponibles. El producto final es un artículo científico con la colección de herramientas.

Medir el impacto de objetos paralelos en aplicaciones. Se procederá a tomar programas de *benchmark* producidos por centros de computación extrema en el sistema de laboratorios nacionales de Estados Unidos. Luego, se procederá a hacer una comparación de una colección representativa de tales programas al correrlos en la supercomputadora *Kabré* del Centro Nacional de Alta Tecnología. Los resultados de este estudio ayudarán a entender el impacto de cada una de las ventajas de los objetos paralelos en los programas científicos.

Proponer un diseño del sistema de objetos paralelos en un lenguaje de alto nivel. Este objetivo analizará la forma en que el sistema de objetos paralelos para un lenguaje de alto nivel (Python) debe organizarse. Así mismo, ofrecerá una interfaz clara de la forma en que los objetos paralelos se utilizarán en el lenguaje. El resultado de esta actividad es un diseño con los principales componentes y la interfaz del lenguaje.

Construir un prototipo con funcionalidad básica. Este objetivo se logrará al implementar las estructuras de datos básicas para manipular objetos paralelos en Python. El entregable consiste en un prototipo que sea capaz de crear un arreglo de objetos y pueda entregar mensajes a los objetos.

Resultados

Investigación

Tradicionalmente, el énfasis de la computación de alto rendimiento (HPC) los centros de datos y las aplicaciones han estado en el desempeño. Sin embargo, se anticipa que la generación futura de sistemas de supercomputación enfrentarán grandes desafíos en confiabilidad, administración de energía y variaciones térmicas. Se requieren soluciones disruptivas para optimizar el rendimiento en la presencia de estos desafíos. Un sistema de tiempo de ejecución inteligente y paralelo que es parte de cada trabajo, y que interactúa con un administrador de recursos adaptativo para la máquina como un todo, es clave para superar los desafíos de la próxima generación centros de datos de supercomputación. Se ha demostrado que un sistema de tiempo de ejecución inteligente y adaptable puede:

- Mejorar la eficiencia en un entorno con restricciones de energía.
- Aumentar el rendimiento con algoritmos de equilibrio de carga.
- Controlar la fiabilidad de los supercomputadores con sustanciales variaciones térmicas.
- Configurar componentes de hardware para operar dentro de restricciones de potencia y / o para ahorrar energía.

Aunque estas líneas de investigación se desarrollaron en aislamiento, indican que los sistemas de tiempo de ejecución inteligente tienen una gran potencial para superar las barreras hacia la exaescala computacional. Lo que la comunidad de HPC no tiene es un sistema integrado que combina investigaciones pasadas en un solo sistema que optimiza en múltiples dimensiones. Se propuso un diseño integral basado en objetos paralelos (resumido en la Figura 5) en el que el administrador de recursos del centro de datos:

- Interactúa dinámicamente con los sistemas de tiempo de ejecución individuales de trabajos.
- Optimiza tanto el rendimiento como el consumo de energía.
- Opera en un entorno con fallas del sistema bajo restricciones proporcionadas por usuarios o administradores.

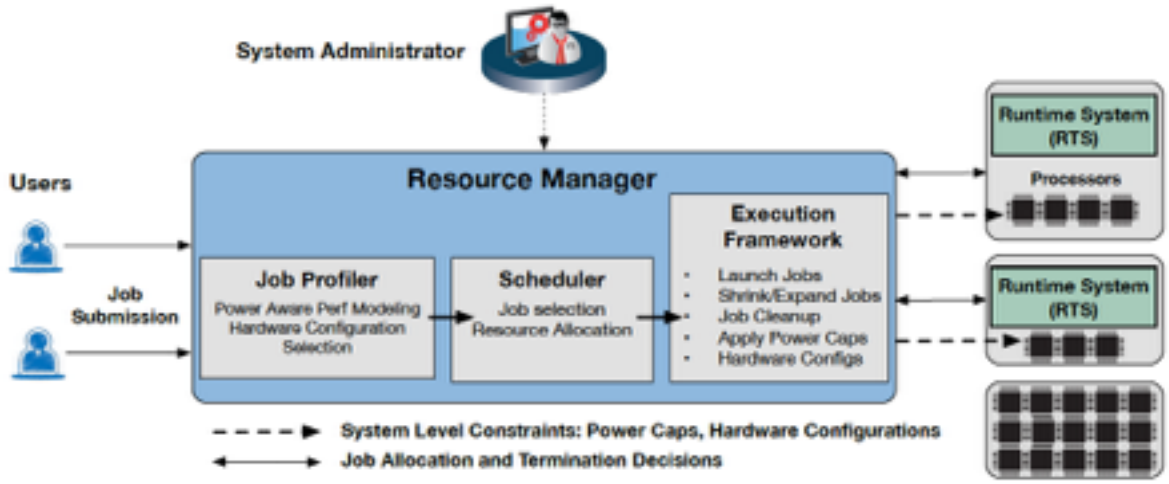

Figura 5. Diseño de una plataforma de computación de alto rendimiento basada en objetos paralelos para decrementar el costo energético y aumentar la resiliencia.

Los objetos paralelos ofrecen entonces un modelo atractivo para desarrollar sistemas en lenguajes de alto nivel que permitan el desarrollo de aplicaciones científicas o ingenieriles. El lenguaje Python fue escogido como el destino para esta implementación. El lenguaje de programación Python es ampliamente utilizado en informática. Cinco rankings sobre la popularidad de la programación idiomas con diferentes metodologías lista Python en la posición 1, 4, 2, 3 y 3, respectivamente. Esas clasificaciones consideraron la cantidad de resultados devueltos por búsquedas web, proyectos Github, preguntas etiquetadas en StackOverflow, abrir puestos de trabajo, entre otras variables. En el 2015 aproximadamente el 10% de todos los trabajos de computación en Texas Advanced Computing Center eran aplicaciones de Python. También, existen módulos específicos para una gran diversidad de disciplinas. Debido a esta sólida base de código y facilidad de uso, Python es comúnmente el primer idioma aprendido por ingeniería, estudiantes de ciencias básicas y aplicadas.

La comunidad de Python generó una plétora de proyectos que proporcionan paralelismo a través módulos, llamadas a funciones, envío de trabajos, anotaciones, compiladores *jit* y sistemas de tiempo de ejecución. Teniendo en cuenta su popularidad, potencial de crecimiento y ausencia de estándar de facto para computación paralela, proporcionando paralelismo en el lenguaje de programación Python es una investigación prometedora campo. Un paso importante para proponer soluciones efectivas es entender qué estrategias ya se han implementado y sus resultados. Se desarrolló una recolección de proyectos que apuntan a proporcionar el paralelismo en Python.

TABLE I
CLASSIFICATION OF TOOLS THAT PROVIDE PARALLELISM IN THE PYTHON PROGRAMMING LANGUAGE.

Project	Execution strategy	Parallel paradigm	Vector data oriented	Language support	Code modifications	Parallel platform	Latest release
Bohrium [14]	Interpreted	Data	Yes	Full, Python 2	None	SMP, GPU, Clusters	0.3, Apr-2016
PyStream [15]	Compiled	Data	Yes	Subset, Python 2	None	GPU	0.1, Jul-2011
Dask.array [16]	Interpreted	Data	Yes	Full, Python 3	FunCall	SMP, Clusters	0.13.0, Jan-2017
PupyMPI [17]	Interpreted	MsgPsg	No	Full, Python 2	FunCall	SMP, Clusters	0.9.5, May-2011
Pupy [18]	Interpreted	Task	No	Full, Python 2	JobSub	SMP, Clusters	1.0.8, Nov-2014
GAiN [19]	Binary binding	Data	Yes	Full, Python 2	FunCall	Clusters	1.0, 2009
Global Arrays [20]	Binary binding	Data	Yes	Full, Python 2	FunCall	Clusters	5.5, Aug-2016.
mpi4Py [21]	Binary binding	MsgPsg	No	Full, Python 2-3	FunCall	SMP, Clusters	2.0.0, Oct-2015
Pythran [22]	Compiled	Data	Yes	Subset, Python 3	Annotations	SMP	0.7.6.1, Jul-2016
ASP [23]	Binary binding	Data, Task	No	Full, Python 2	JobSub	SMP, GPU	0.1.3.1, Oct-2013
Disco4py [24]	Interpreted	Data, Task	No	Full, Python 2-3	JobSub	SMP, Clusters	1.2, Jan-2015
PMI [25]	Interpreted	Data	No	Full, Python 2-3	FunCall	SMP, Clusters	1.0, Dec-2009
Jit4OpenCL [26]	Compiled	Data	Yes	Full, Python 2	Annotations	SMP, GPU	1.0, 2010
MRS [27]	Interpreted	MapRed	No	Full, Python 2-3	FunCall	Clusters	0.9, Nov-2012
Pydron [28]	Interpreted	Task	No	Subset	Annotations	Clusters	–
CoArray [29]	Interpreted	Data	Yes	Full, Python 2	FunCall	Clusters	2004
PyCuda, PyOpenCL [30]	Binary binding	Data	Yes	Full, Python 2-3	FunCall	SMP, GPU	2016.2, Oct-2016
SCOOP [31]	Interpreted	Task	No	Full, Python 2-3	JobSub	SMP, Clusters	0.7.1.1, Ago-2015
DistArray [32]	Interpreted	Data	Yes	Full, Python 2-3	JobSub	SMP, Clusters	0.6, Oct-2015
Dispy [33]	Interpreted	Data, MapRed	No	Full, Python 2-3	JobSub	SMP, Clusters	4.6.17, Sep-2016
IpyParallel [34]	Interpreted	Data, Task	No	Full, Python 2-3	JobSub	SMP, Clusters	5.3.0, Oct-2016
PyRo [35]	Interpreted	MsgPsg	No	Full, Python 2-3	Annotations, FunCall	Clusters	4.50, Nov-2016
Parallel python [36]	Interpreted	Task	No	Full, Python 2-3	JobSub	SMP, Clusters	1.6.5, Jul-2016
JUG [37]	Interpreted	Task	No	Full, Python 2-3	Annotations, FunCall	SMP, Clusters	1.3.0, Nov-2016
Multiprocessing [38]	Interpreted	Task, Data	No	Full, Python 2-3	FunCall	SMP, Clusters	3.6, Jul-2016
Copperhead [39]	Binary binding	Data	Yes	Subset, Python 2	Annotations	GPU	2013
Celery [40]	Interpreted	Task	No	Full, Python 2-3	Annotations, FunCall	SMP, Clusters	4.0.0, Nov-2016
Disco [41]	Interpreted	MapRed	No	Full, Python 2	Annotations, FunCall	SMP, Clusters	0.5.4, Oct-2014
Spark [42]	Binary binding	Task	No	Full, Python 2-3	FunCall	Clusters	2.0.2, Nov-2016
Theano [43]	Binary binding	Data	Yes	Full, Python 2-3	FunCall	SMP, GPU, Clusters	0.8.2, Ago-2016
Numba [44]	Compiled	Data	Yes	Full, Python 2-3	Annotations	SMP, GPU	0.29.0, Oct-2016
Joblib [45]	Interpreted	Task	No	Full, Python 2-3	JobSub, Annotations	SMP	0.10.3, Oct-2016
Hadoopy [46]	Binary binding	MapRed	No	Full, Python 2	JobSub	Clusters	0.5.0, Jun-2012
PyMW [47]	Interpreted	Task	No	Full, Python 2	FunCall	Clusters	0.4, Jun-2010
Pyfora [48]	Compiled	Data	No	Subset, Python 2	None	Clusters, SMP	0.5.8, Set-2016

Figura 6. Colección de herramientas de programación paralela en Python y sus características.

La tabla en la Figura 6 ofrece una vista a las herramientas de programación paralela en Python. La tabla fue tomada de [PYTHONPAR]. Las herramientas se clasificaron de acuerdo a cómo se ejecuta el código, cómo el paralelismo está expuesto, si los vectores son la estructura principal de datos, soporte de versiones de Python, cómo se debe modificar el código para que se ajuste a la herramienta, a qué plataforma pertenece la herramienta y la última fecha de lanzamiento. Las características más comunes son: usar el intérprete de Python ejecutar el código o usar código externo (enlace binario), ofreciendo paralelismo de datos o paralelismo de tareas, si el paralelismo de datos se ofrece en general la estructura de datos principal serán vectores. La mayoría de los proyectos admiten la especificación completa del lenguaje. El paralelismo es expuesto como llamadas a funciones o envío de trabajos explícito. La orientación SMP y plataforma de clusters es común, este último más que el primero. La Figura 7 presenta un gráfico de co-ocurrencia de características en la colección de herramientas.

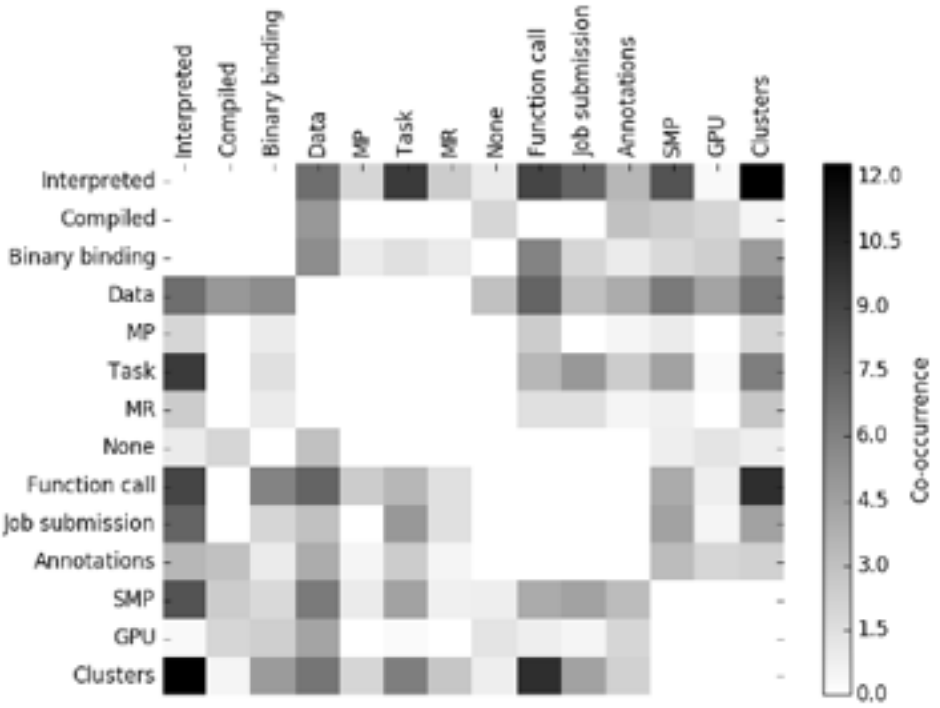

Figura 7. Co-ocurrencia de características en la colección de herramientas de programación paralela en Python.

El diseño de una herramienta que integre los mundos de objetos paralelos y Python debe seguir alguno de las siguientes estrategias:

- Implementación pura en Python. Todas las funcionalidades de Charm++ se implementan directamente en Python y se ofrecen como llamadas a una biblioteca o extensiones del lenguaje. Esto hace que la interfaz sea 100% compatible con Python.
- Compilación de código Python a Charm++. El código Python, posiblemente con extensiones para permitir expresar mejor las características de los objetos paralelos, es migrado a Charm++, donde se enlaza con el código de la biblioteca y es ejecutado como un programa en el lenguaje C++.

- Un compromiso entre las dos opciones anteriores, lo que significa buscar un punto intermedio en el que código Python es utilizado junto con código Charm++.

Publicaciones

- Bilge Acun, Akhil Langer, Esteban Meneses, Harshitha Menon, Osman Sarood, Ehsan Totoni and Laxmikant V. Kalé. *Power, Reliability, and Performance: One System to Rule them All*. IEEE Computer, 2016.
- Guillermo Cornejo-Suárez y Esteban Meneses, *Parallel Programming Tools in Python*, III Jornadas Costarricenses de Investigación en Computación e Informática, Cartago, Costa Rica. Agosto, 2017.

Discusión y conclusiones

- El modelo de objetos paralelos ofrece una abstracción prometedora para la implementación de sistemas de computación paralela en lenguajes de alto nivel. Este modelo ofrece la posibilidad de crear los objetos que representan las entidades fundamentales en el dominio científico en el que se circunscribe la simulación científica.
- El lenguaje de programación Python representa una buena oportunidad para desarrollar una implementación de objetos paralelos. Python es un lenguaje orientado a objetos que naturalmente podría incorporar el modelo de objetos paralelos. También, Python ofrece mecanismos para conectar otras bibliotecas y desarrollar aplicaciones más complejas.
- Los diseños de objetos paralelos en Python arrojan una gama interesante de posibilidades por explorar. El primer diseño consiste en reimplementar Charm++ en Python, lo que incrementa el tiempo de desarrollo, pero permite una interfaz pura en Python. El segundo diseño, supone una compilación de Python a Charm++, que reduce el tiempo de desarollo (consiguiendo un compilador apropiado), pero entorpece los procesos de depuración de código. El tercer diseño es un punto intermedio donde se pueden mezclar los dos mundos de Python y Charm++.

Recomendaciones

- La incorporación de estudiantes asistentes es vital para el cumplimiento de los objetivos. Las estructuras de investigación en los centros científicos del primer mundo son jerárquicos, donde el profesor ejerce un rol de liderazgo y administración de un grupo de trabajo. Esta jerarquía debería incorporar: estudiantes de doctorado (principalmente), estudiantes de maestría, estudiantes asistentes de grado, investigadores y otros invitados.
- Un elemento transformador en la investigación es la colaboración internacional. Mantener contactos con colegas de otras universidades no solo incrementa la productividad científica (al incrementar el equipo de trabajo), sino que refina la autocrítica y la retroalimentación de las ideas. Nuestros colegas en otros países sirven como punto de comparación para evaluar el estado de nuestras capacidades de investigación.
- Contar con financiamiento complementario, como el Fondo de Desarrollo de la Unidad (FDU), es una ayuda de muy alto impacto. Con fondos del FDU se hizo la compra de equipo de cómputo para realizar demostraciones, particularmente en la presentación de la charla *Supercomputación para todos*. La disponibilidad de estos fondos, aunque de

reducido presupuesto, permiten la adquisición rápida de elementos para la investigación y potencian el desarrollo de la misma.
- Establecer convenios para acceso a equipo es recomendable para facilitar el desarrollo de proyectos de investigación. En particular, el acceso al cluster computacional del CeNAT hizo que los experimentos fueran mucho más sencillos de correr (por el entorno ya instalado en el CeNAT) y con capacidades mucho mayores a las disponibles en el Tec.

Agradecimientos

Luis Guillermo Cornejo es un estudiante del programa de Maestría en Computación con énfasis en Ciencia de la Computación del Instituto Tecnológico de Costa Rica. Luis Guillermo trabajó en el proyecto, principalmente en la comparación de herramientas de computación paralela en Python.

Esta investigación contó con el apoyo de una asignación computacional en la supercomputadora Kabré del Centro Nacional de Alta Tecnología de Costa Rica.

Referencias

[CHANGA] Pritish Jetley, Filippo Gioachin, Celso Mendes, Laxmikant Kale, Thomas Quinn. Massively Parallel Cosmological Simulations with ChaNGa (IPDPS, 2008).
[CHARM++] Laxmikant V. Kale, Abhinav Bhatele, Parallel Science and Engineering Applications: The Charm++ Approach (CRC Press, 2013).
[CHARMEN] Osman Sarood, Akhil Langer, Abhishek Gupta, Laxmikant Kale. Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget (Supercomputing, 2014).
[CHARMFT] Esteban Meneses, Xiang Ni, Gengbin Zheng, Celso Mendes, Laxmikant Kale. Using Migratable Objects to Enhance Fault Tolerance Schemes in Supercomputers (IEEE Transactions on Parallel and Distributed Systems, 2014)
[DA] Distributed Array Protocol.
Sitio web: http://distributed-array-protocol.readthedocs.org/en/rel-0.10.0/protocol.html
[GA] Manojkumar Krishnan, Bruce Palmer, Abhinav Vishnu, Sriram Krishnamoorthy, Jeff Daily, and Daniel Chavarria. The Global Arrays User Manual (2012).
[GRA03] Ananth Grama et al, Introduction to Parallel Computing, 2nda edición (Addison-Wesley, 2003).
[HAG10] Georg Hager y Gerhard Wellein, Introduction to High Performance Computing for Scientists and Engineers (Chapman & Hall, 2010).
[KIR12] David Kirk y Wen-mei Hwu, Programming Massively Parallel Processors, 2nda edición (Morgan Kaufmann, 2012).
[LIN08] Calvin Lin y Larry Snyder, Principles of Parallel Programming (Addison-Wesley, 2008).
[MAT13] Timothy G. Mattson, Beverly A. Sanders y Berna L. Massingill, Patterns for Parallel Programming (Addison-Wesley, 2013).
[McC08] Michael McCool, Arch D. Robison y James Reinders, Structured Parallel Programming (Morgan Kaufmann, 2008).
[MPI] William Gropp, Ewing Lusk, Anthony Skjellum. Using MPI: Portable Parallel Programming with the Message-Passing Interface , 3era edición (The MIT Press, 2014).
[MPI4PY] Biblioteca MPI4Py. Sitio web: https://pypi.python.org/pypi/mpi4py

[NAMD] James Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert Skeel, Laxmikant Kale, Klaus Schulten. Scalable Molecular Dynamics with NAMD (Journal of Computational Chemistry, 2005).
[OPENATOM] Ramkumar Vadali, Yan Shi, Sameer Kumar, Laxmikant Kale, Mark Tuckerman, Glenn Martyna. Scalable Fine-Grained Parallelization of Plane-Wave-Based ab initio Molecular Dynamics for Large Supercomputers (Journal of Computational Chemistry, 2004).
[PARALLEX] Hartmut Kaiser, Maciej Brodowicz, Thomas Sterling: ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications (International Conference on Parallel Processing Workshops, 2009).
[PAC11] Peter Pacheco, An Introduction to Parallel Programming (Morgan Kaufmann, 2011).
[PYTHON] Lenguaje de programación Python. Sitio web: https://www.python.org
[PYTHONPAR] Guillermo Cornejo-Suárez y Esteban Meneses, *Parallel Programming Tools in Python*, III Jornadas Costarricenses de Investigación en Computación e Informática, Cartago, Costa Rica. Agosto, 2017.
[QUI03] Michael Quinn, Parallel Programming in C with MPI and OpenMP (McGraw-Hill, 2003).