Title: Automatic task-based parallelization for OO languages
Supervisor: Bérenger Bramas (berenger.bramas@inria.fr)

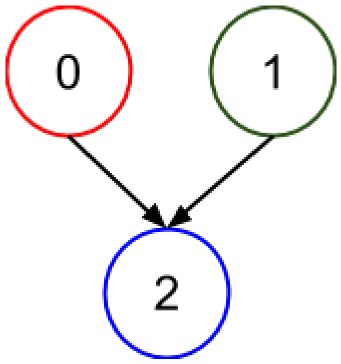
Starting data: between February and April 2019
Duration: 5 or 6 months
Location: ICPS team (Inria CAMUS) in the ICUBE laboratory, Illkirch Graffenstaden, France

---

Context:
The CAMUS's research team focus on automatic parallelization and optimization, profiling, modeling, and compilation. The team has increasing interests for the approaches used and enhanced in the high-performance community. In this field, the task-based method has gained popularity because it allows parallelizing with an abstraction of the hardware by delegating task distribution and load balancing to the dynamic schedulers. Consequently, all the aspects related to the TB parallelization are active research topics but parallelizing a code is still done by hand.

Objectives:
The project aims in having an automatic task-based parallelization system for C++ applications using the sequential task-flow (STF) model. In this model, a sequential code (i.e. a code that is originally not designed to run in parallel) is annotated to decide what sections to transform into tasks and how these ones access the data. From this description, a runtime system can build a graph of tasks and execute them in parallel, while respecting the data dependencies.

Sequential code	STF code	Graph of tasks

Matrix A, B, C;
 	
A = GetMatrix();
B = GetMatrix();
 	
C = A × B;
	Matrix A, B, C;
	
	#task write(A) // Task 0
	{
	    A = GetMatrix();
	}
	#task write(A) // Task 1
	{
	    B = GetMatrix();
	}
	
	#task write(C) read(A,B) // Task 2
	{
	    C = A × B;
	}
		

The transformation from an STF code to a graph of tasks is done by a runtime system. The current project will focus on the transformation from a sequential code to an STF code. To do so, a source-to-source transformation of sequential C++ source code will insert tasks and data accesses. The first approach will consider that one function/method is one task and will manage the data accesses at "object" level (one object == one data handle). Then, the research work will focus on fighting the successive pitfalls that will arise and increasing the performance/degree of parallelism.

Sequential code	Possible simple STF code (?)	Possible advanced STF code (?)
LogManager GlobalLog;	LogManager GlobalLog;	LogManager GlobalLog;
class A{	class A{	class A{
public:	public:	public:
void fa(){	void fa(){	void fa(){
GlobalLog(“A.fa”);	#task_group{	#extend task_group{
}	#task write(GlobalLog)	#task
};	GlobalLog(“A.fa”);	atomic_write(GlobalLog)
	}	GlobalLog(“A.fa”);
class B{	}	} // No sync here
public:	};	}
void fb(){		};
A a;	class B{	
a.fa();	public:	class B{
}	void fb(){	public:
void fb2(const B& b){	A a;	void fb(){
}	#task_group{	#extend task_group{
};	#task write(a)	// a is local and only
	a.fa();	// used in this task
int main(){	}	#task {
B b1, b2;	}	A a;
b1.fb();	void fb2(const B& other){	a.fa();
b2.fb();	}	}
b1.fb2(b2);	};	} // No sync here
return 0;		}
}	int main(){	void fb2(const B& other){
	B b1, b2;	}
	#task_group{	};
	#task write(b1)	
	b1.fb();	int main(){
	#task write(b1)	B b1, b2;
	b2.fb();	#task_group{
	#task write(b1) read(b2)	#task write(b1)
	b1.fb2(b2);	b1.fb();
	}	#task write(b1)
	return 0;	b2.fb();
	}	#task write(b1) read(b2)
		b1.fb2(b2);
		} // will sync with tasks
		// created in fb()
		return 0;
		}

We provide here a non-exhaustive list of possible milestones:

- How to manage the granularity to enable/disable tasking at runtime
- How/when to split a function/loop into tasks
- How to detect patterns for commutative data accesses or speculative execution
- How/when to remove/delay/regroup synchronizations
- How to schedule an automatically parallelized application

The significant results from researches on automatic parallelization will give interesting hints. The validation of the proposed solutions will be done on large open-source C++ projects that are currently sequential.

Technology: The source-to-source transformation system will be based on clang-LLVM. It will be developed on a modern Gitlab and proposed to the community with an open-source license. We will rely on our inhouse runtime system SPETABARU to manage and execute the tasks.

Diploma: Internship for last year of an M.S. in computer science (BAC+5). University or engineering school with orientation in research.

Skills:

- Proactive, high interest in solving problems, interest in learning clang-LLVM
- C++ (17)
- Knowledge of compilation would be really appreciated
- Knowledge of parallel programming/computing is a plus

Benefit: The legal compensation will be offered (approx. 577 euros per month)

You will find a great working environment with flexible schedule, nice atmosphere and opportunities to give the best of yourself.