Handel-C++

Adding Syntactic Support To C++

Alex Cole
University of Leicester

CPA 2012, Dundee
26th – 29th August

Alistair A McEwan
University of Leicester

Geoff Mainland
Microsoft Research

```
par
{
        Process2();
        Process3();
}
```

```
#define par \
        for (
                _CSPPP_CURRENT_THREAD._ParStart();
                _CSPPP_CURRENT_THREAD._LoopCheck();
                _CSPPP_CURRENT_THREAD._LoopEnd())

__declspec((thead)) CSPPP
        _CSPPP_CURRENT_THREAD;
```

```
class CSPPP
{
	std::stack<CSPPPScope *> m_scope;
	bool m_loop;
	void _ParStart()
	{
		m_scope.push(new ParScope());
		m_loop = true;
	};
}
```

```
bool _LoopCheck()
{
	return m_loop;
}

void _LoopEnd()
{
	CSPPPScope * run = m_scope.pop();
	run->_Run();
	m_loop = false;
	delete run;
}
```

```
par
{
        proc Process2();
        proc Process3();
}
```

```
#define proc \
        _ CSPPP_CURRENT_THREAD = new

CSPPP & operator=(_Process * p)
{
        m_scope.peek()->Add(p);
        return *this;
}
```

```
class ParScope
{
        std::list<_Process *> m_children;
        void Add(_Process * p)
        {
                m_children.add(p);
        }
}
```

```
void Run()
{
	// Not C++ (for).
	foreach (_Process * c : m_children)
	{
		_ CSPPP_CURRENT_THREAD .Schedule(c);
	}
	_Yield();
}
```

```
par
{
        proc Process2();
        seq
        {
                proc Process3();
                proc Process4();
        }
}
```

```
#define par \
        for (
                _CSPPP_CURRENT_THREAD._SeqStart();
                _CSPPP_CURRENT_THREAD._LoopCheck();
                _CSPPP_CURRENT_THREAD._LoopEnd())
```

```
class SeqScope
{
	void Add(_Process * p)
	{
		p->_Run();
		delete p;
	}
	void Run() {}
}
```

```
chan int c1;
par
{
        proc Process2(c1);
        seq
        {
                proc Process3();
                proc Process4(c1);
        }
}
```

```
Chan<int> c1;
par
{
        proc Process2(c1);
        seq
        {
                proc Process3();
                proc Process4(c1);
        }
}
```

```
int var = *chan;
*chan = 42;

class Chan
{
        ChanAccess & operator* ()
        {
                return m_access;
        }
}
```

```
class Chan<T>
{
        ChanAccess<T> & operator* ()
        {
                return m_access;
        }
}
```

```
class ChanAccess<T>
{
	operator T()
	{
		return ReadChannel(m_parent);
	};
	ChanAccess<T> & operator=(T & d)
	{
		WriteChannel(m_parent, d);
		return *this;
	}
}
```

- No working scheduler
- Crashes randomly (possibly data races)
- Alternate existing backend