



# Productivity Gains in Erlang



A Heriot-Watt and Motorola Production

Commercial Users of Functional Programming  
Freiburg, Germany, October 4<sup>th</sup> 2007



Erlang Training and Consulting Ltd  
[www.erlang-consulting.com](http://www.erlang-consulting.com)

Jan Henry Nyström

[jann@macs.hw.ac.uk](mailto:jann@macs.hw.ac.uk)

[jan@erlang-consulting.com](mailto:jan@erlang-consulting.com)

Phil Trinder

[trinder@macs.hw.ac.uk](mailto:trinder@macs.hw.ac.uk)

David King

[david.king@praxis-his.com](mailto:david.king@praxis-his.com)

## Project

- High-Level Techniques for Distributed Telecoms Software
- EPSRC (UK Govt) Project, Dec 2002 - Feb 2006
- Collaboration between
  - Motorola UK Labs
  - Heriot-Watt University

## Project

- High-Level Techniques for Distributed Telecoms Software
- EPSRC (UK Govt) Project, Dec 2002 - Feb 2006
- Collaboration between
  - Motorola UK Labs
  - Heriot-Watt University
- **Aim:** Produce scientific evidence that high-level distributed languages like Erlang or Glasgow distributed Haskell (GdH) can improve distributed software robustness and productivity.

## Erlang Comparisons

- A number of sequential comparisons
  - *Computer Language Shootout*
- Very few distributed system comparisons published!
- Ulf Wiger [Wiger01] reports
  - Erlang systems have between 4 and 10 times less code than C/C++/Java/PLEX systems
  - Similar error rates/line of code, similar productivity rates
  - No direct comparative measurements

## Research Questions: Potential Benefits

- RQ1: Can robust, configurable systems be readily developed?
- Resilience to extreme loads
- Availability in the face of hardware & software failures
- Dynamic reconfigurability on available hardware
  
- RQ2: Can productivity and maintainability be improved?
- How do the sizes of the C++ and Erlang components compare & what language features contribute to size differential?

## Research Questions: Feasibility

High-level distributed languages:

- Abrogate control of low-level coordination aspects, so
  - *RQ3: can the required functionality be specified?*
- typically pay space and time penalties for their automatic coordination management.
  - *RQ4: can acceptable performance be achieved?*
- RQ5: What are the costs of interoperating with conventional technology?
- RQ6: Is the technology practical?

## Research Strategy

- Reengineer telecoms applications in GdH and Erlang
  - *Dispatch Call Controller [NTK04,NTK05]*
  - *Data Mobility component*
- Compare high-level and Java/C++ implementations for
  - *Performance*
  - *Robustness*
  - *Productivity*
  - *Impact of programming language constructs*

## Data Mobility Component (DM)

### First Product Application

- Communicates with Motorola mobile devices
- 3000 lines of C++
- Uses 18,000 lines of Motorola C library functions
- Has a transmitter, a receiver and 2 message types
- Interacts with 5 other components of the system

## Despatch Call Controller (DCC)

### Second Application

- Handles mobile phone calls
- A process manages each call
- Scalable with multiple servers

## Two Erlang DM Implementations

1. Pure Erlang DM
  2. Erlang / C DM
    - Reuses some C DM libraries & drivers
- Both interoperate with a test harness
  - Combines
    - *Unix Processes*
    - *Erlang processes*
    - *C Threads (Erlang/C DM)*

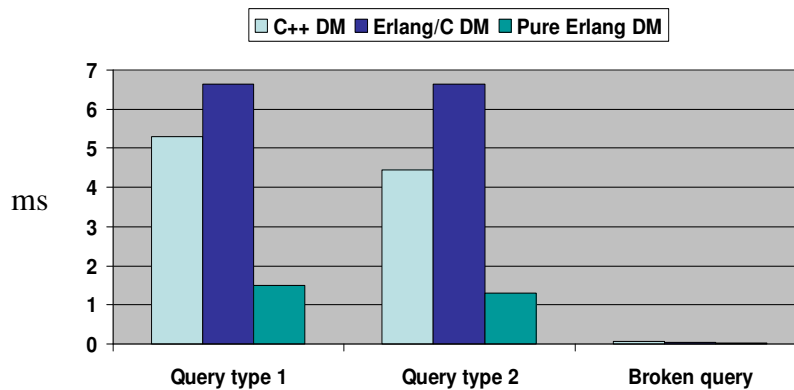
## RQ3 Performance 1: Throughput

Platform: 167MHz, 128Mb Sun Ultra 1, SunOS 5.8

C++ DM	Erlang/C DM	Pure Erlang DM
480	230	940

- Maximum DM Throughput at 100% QoS
- Pure Erlang DM is **twice as fast** as C++ DM
  - Better process management and marshalling
- Erlang/C DM is **½ speed** of C++ DM
  - Still meets nominal throughput

## RQ3 Performance 2: Roundtrip Times

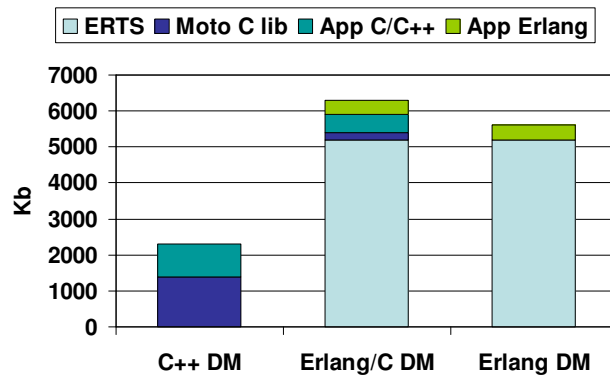


- Pure Erlang is approximately **3 times faster**
- Erlang/C is **26% - 50% slower**

## Performance Analysis

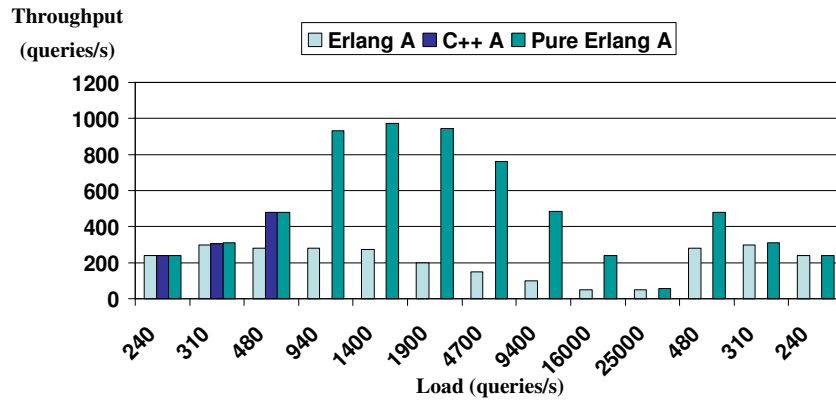
- Pure Erlang DM is faster due to fast lightweight process management
- Erlang/C is slower due to additional communication to C components

## Performance 3: Memory Residence



- Erlang DMs use **170% more memory**
- Erlang runtime sys (ERTS) has a fixed size
  - Would be a smaller % of a larger application

## RQ1 Robustness 1: Resilience



## RQ1 Robustness 1: Resilience

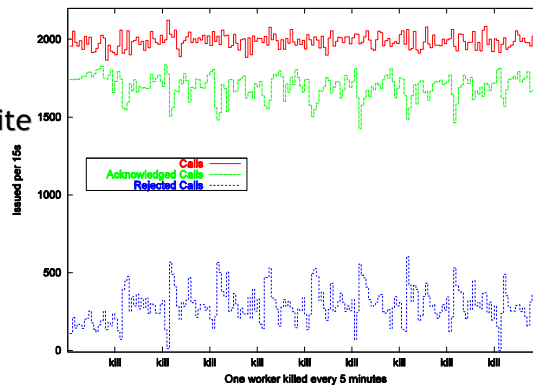
When overloaded:

- C++ DM fails catastrophically
- Pure Erlang & Erlang/C DMs:
  - Throughput degrades
    - Never completely fails
    - Handling 48 q/s at peak load (25000q/s)
    - Recovers automatically after load drops

## Robustness 2: Availability

Erlang DCC system:

- Remains available despite repeated hardware and software failures
- Performance doesn't degrade with repeated failures

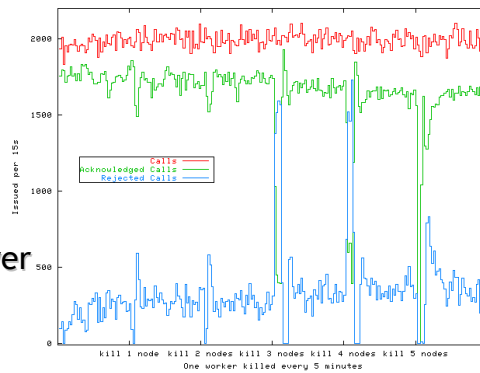


DCC Throughput with Repeated Failures

## Robustness 2: Availability

Erlang DCC system:

- Resists the simultaneous failure of multiple components
- When more components fail throughput drops lower & recovery takes longer

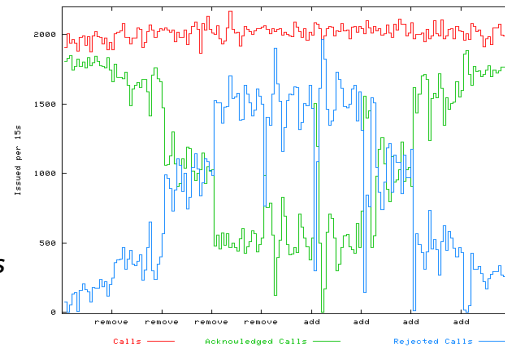


5-processor DCC with Multiple Failures

## Robustness 3: Dynamic Configurability

Erlang DCC system:

- Dynamically adapt to the available hardware resources
- 5 processor system
  - Remove a processor 4 times
  - Add a processor 4 times



DCC Throughput  
with Varying Numbers of Processors

## RQ2: Productivity & Maintainability

- Shorter programs are
- Faster to develop
- Contain fewer errors [Wiger01]
- Easier to maintain
  
- The metric we use is Source Lines Of Code (SLOC)

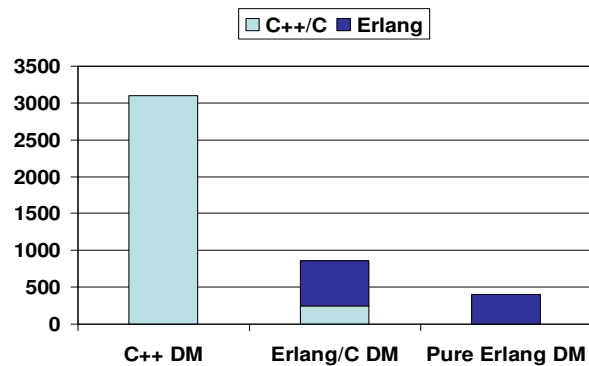
## Productivity: Source Code Sizes

Lang.	C/C++	Erlang	Total	Lang.	C++	IDL	Erlang	Total
C++	3101		3101	C++	14.8K	83		14.9K
Erl./C	247	616	863	Erlang			4882	4882
Erlang		398	398	DCC Implementations				

DM Implementations

- Erlang DCC & DM are less than 1/3rd of size of C++ version
- Consistent with Wiger & folklore

## Productivity: DM Source Code Sizes

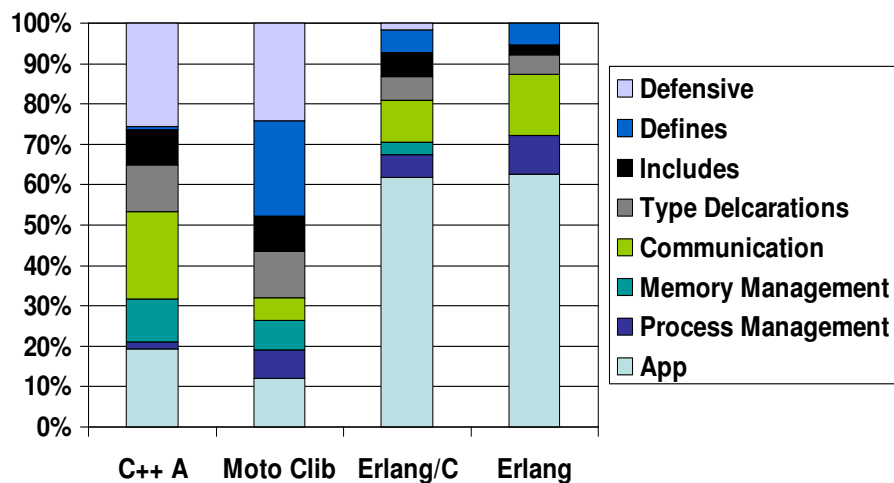


- Erlang/C DM is 1/3rd of the size of the C++ DM
- Pure Erlang DM is 1/7th of the size of the C++ DM
- Erlang/C DM is 1/18th of the size of the C++ DM + libraries

## Reasons for difference in Code Size

- Erlang programmers can
  - Rely on fault tolerance and code for the successful case
  - 27% of C++ DM code is defensive
- And have to their use
  - Automatic memory management (11% of C++ DM code)
  - High-level communication (23% of C++ DM code)
  - Telecom design pattern libraries

## DM Code Breakdown



## Code Difference Example

```

void DataMobiRxProcessor::ProcessMsgRetStatusVer(void)
MSG_PTR      msg_buf_ptr;
MM_DEVICE_INFO_MSG *msg_ptr;
RETURN_STATUS ret_status;
UINT16      msg_size;
    
```

**C++**

```

// Determine size of ICI message
msg_size = sizeof( MM_DEVICE_INFO_MSG);

// Create ICI message object to send to DMTX so it sends a Device Info
// message to VLR and HLR clients
IciMsg ici_msg_object( MM_DEVICE_INFO_OPC, ICI_DMTX_TASK_ID, msg_size);

// Retrieve ICI message buffer pointer
msg_buf_ptr = ici_msg_object.getIciMsgBufPtr();

// Typecast pointer from (void *) to (MM_DEVICE_INFO_MSG *)
msg_ptr = (MM_DEVICE_INFO_MSG *)msg_buf_ptr;

// Populate message buffer
SET_MM_DEVICE_INFO_DEVICE_TYPE( msg_ptr, SERVER);
SET_MM_DEVICE_INFO_NUM_VER_SUPPORTED( msg_ptr, NUM_VER_SUPPORTED);
SET_MM_DEVICE_INFO_FIRST_SUP_PROTO_VERS( msg_ptr, PROTO_VERSION_ONE);

// Send message to the DMTX task
ret_status = m_ici_io_ptr->send(&ici_msg_object);

// Check that message was sent successfully
if (ret_status != SUCCESS)
{
    // Report problem when sending ICI message
    sz_err_msg_MAJOR_SZ_ERR_MSG_ERR_OPCODE_FILE_LINE
    "DataMobiRxProcessor processUnsupVer: failure sending "
    "device info message to DMTX");
}
    
```

**Erlang**

```
sz_dme_dmtx.cast(device_info)
```

## Erlang DCC Reusability

Part	SLOC	No. Modules	Percentage
<b>Reusable Platform</b>	2994	26	<b>61%</b>
<b>Specific Services</b>	147	1	3%
<b>Testing &amp; Statistics</b>	1741	11	36%

- Considerable potential for reuse

## Summary

- Investigated high-level distributed language technology for telecoms software
- Reengineered two telecoms components in Erlang
- Measured & compared the Erlang & C++ components

## RQ1: Robust and Configurable Systems

- **Improved resilience:**
  - Erlang DM and DCC sustain throughput at extreme loads
  - Automatically recover when load drops
  - C++ DM fails catastrophically (predict C++/CORBA DCC would)
- **Improved availability:**
  - Erlang DCC recovers from repeated & multiple failures
  - Predict C++/CORBA DCC would fail catastrophically
- **Dynamic Reconfiguration**
  - Erlang DCC can be dynamically reconfigured to available hardware
  - C++/CORBA DCC can also be dynamically reconfigured using CORBA
- **Potential for hot-code loading (not demonstrated)**

## RQ2: Productivity and Maintainability

- **Erlang DM and DCC:**
  - *Less than 1/3<sup>rd</sup> size of C++ implementation*
  - *Erlang DM 1/18<sup>th</sup> size of C++ DM with libraries*
  - *Good reusability*
- **Reasons:**
  - *Code for successful case - saves 27%*
  - *Automatic memory management - saves 11%*
  - *High-level communications - saves 23%*
  - *Telecom design pattern libraries*

## Distributed Functionality

- **Even though Erlang abstracts over low-level coordination, the required DM and DCC functionality is readily specified.**

## RQ4: Performance

### Time:

- Max. throughput at 100% QoS:
  - Pure Erlang DM is *twice as fast* as C++ DM
  - Erlang/C is *½ as fast* as C++ DM , but still exceeds throughput requirements
- Roundtrip times
  - Pure Erlang DM is *three times as fast* as C++ DM
  - Erlang/C is *between 26% and 50% slower* as C++ DM

### Space:

- Pure Erlang and Erlang/C both have **170% greater memory residency** due to (fixed size) 5Mb runtime system

## RQ5: Interoperation Costs

- Erlang DMs interoperate with a C test harness, and Erlang/C DM incorporates C drivers & library functions
- Costs
  - Low space cost: an additional 15% residency
  - High time cost:
    - Erlang/C roundtrip times up to 6 times pure Erlang
    - Erlang/C max. throughput ¼ of pure Erlang
- Potential for incremental re-engineering of large systems

## RQ6: Pragmatics

- Erlang is available on several HW/OS platforms, including the Sun/Solaris DM product platform
- Is supported with training, consultancy, libraries etc.

## Conclusions

- Erlang offers robustness & productivity benefits for distributed telecoms software (RQs 1 & 2)
- High-level distributed languages like Erlang can deliver the required telecoms functionality and performance (RQs 3 & 4)
- Erlang can interoperate with existing technologies and meets pragmatic requirements (RQs 5 and 6)

## Further Information

- Web sites/Seminars
  - Project Site: [www.macs.hw.ac.uk/~dsg/telecoms/](http://www.macs.hw.ac.uk/~dsg/telecoms/)
- References
  - [NTK04] Nystrom J.H. Trinder P.W. King D.J. Evaluating Erlang for Robust Telecoms Software S3S'04, Chicago, July 2004.
  - [NTK05] Nystrom J.H. Trinder P.W. King D.J. Are High-Level languages suitable for Robust Telecoms Software? SafeComp'05, Fredrikstad, Norway, Sept. 2005.
  - [Wiger01] Ulf Wiger, Workshop on Formal Design of Safety Critical Embedded Systems, Munich, March 2001  
[http://www.erlang.se/publications/Ulf\\_Wiger.pdf](http://www.erlang.se/publications/Ulf_Wiger.pdf)